НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет
Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:
Завідувач кафедри
_____ Олександр Коваль
«___»_____2020 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Програмне забезпечення розподілених систем»

спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Веб-сервіс гідроакустичних розрахунків за моделями системи KRAKEN»

Виконав:
студент IV курсу, групи ТВ-61
Войтович Андрій Вікторович _____

Керівник:
Доцент, канд. тех. наук
Варава Іван Андрійович _____

Рецензент:
Доцент, канд. тех. наук
Корольов Андрій Павлович _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки: 121 Інженерія програмного забезпечення

Спеціалізація: Програмне забезпечення розподілених систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр Коваль
(підпис)
”___”_____2020р.

ЗАВДАННЯ

на дипломну роботу студенту

Войтовичу Андрію Вікторовичу
(прізвище, ім’я, по батькові)

1. Тема роботи “Веб-сервіс гідроакустичних розрахунків за моделями системи KRAKEN

керівник роботи Варава Іван Андрійович, доц., к.т.н
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”25” травня 2020р. **№ 1168-с**

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи Мови програмування C#, JavaScript, середовище розробки Microsoft Visual Studio

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Провести аналіз моделей програми KRAKEN. Розробити алгоритми гідроакустичних розрахунків за моделями KRAKEN. Спроектувати веб-сервіс. Реалізувати розроблений алгоритм для використання у веб-сервісі. Створити зручний графічний користувацький інтерфейс для введення вхідних даних та перегляду результатів у вигляді таблиць та графіків.

5. Перелік ілюстративного матеріалу

Актуальність, Завдання роботи, Програма KRAKEN, Структура програми KRAKEN, Засоби розробки, Архітектура системи, Діаграма класів ППІ основного модуля, Користувацький інтерфейс, Таблиця з підрахованими амплітудами нормальних мод на заданих глибинах, Графік залежності величини амплітуди нормальних мод від глибини, Таблиця з підрахованою втратою передачі на обраних глибинах джерела та приймача, Графік залежості втрати передачі від діапазону приймача, Висновки

7. Дата видачі завдання ”11” жовтня 2019 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	11.10.2019	
2.	Вивчення та аналіз задачі	11.10.2019-23.12.2019	
3.	Розробка архітектури та загальної структури системи	03.02.2020-04.03.2020	
4.	Розробка структур окремих підсистем	05.03.2020-12.04.2020	
5.	Програмна реалізація системи	13.04.2020-17.05.2020	
6.	Оформлення пояснювальної записки	18.05.2020-07.06.2020	
7.	Захист програмного продукту	14.05.2020	
8.	Передзахист	10.06.2020	
9.	Захист	16.06.2020	

Студент _____ Войтович А.В.
(підпис) (прізвище та ініціали,)

Керівник роботи _____ Варава І.А.
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Записка містить 47 сторінок, 33 рисунки, 3 додатки, перелік посилань на використані джерела з 17 найменувань.

Метою дипломної роботи є створення веб-сервісу для проведення гідроакустичних розрахунків за моделями KRAKEN.

Об’єктом дослідження є моделі системи KRAKEN та існуючі системи для гідроакустичних розрахунків, побудовані на їх основі.

Результатом роботи стало створення веб-застосунку, який надає користувачу проводити можливість гідроакустичні розрахунки на основі моделей KRAKEN, та аналізувати результуючі дані, використовуючи графічний інтерфейс.

Ключові слова: веб-сервіс, KRAKEN, гідроакустика, C#, ASP.NET Core, JavaScript, React.

ABSTRACT

The thesis in consist of 47 pages. It contains 33 figures, 3 appendixes, bibliography of 17 references.

The purpose of the thesis is to create a web service for hydroacoustic calculations on KRAKEN models.

The object of research is the models of the KRAKEN system and the existing systems for hydroacoustic calculations, built on their basis.

The result was the creation of a web application that allows the user to perform sonar calculations based on KRAKEN models, and analyze the resulting data using a graphical interface.

Keywords: web service, KRAKEN, hydroacoustics, C #, ASP.NET Core, JavaScript, React.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

KRAKEN — програма для гідроакустичних розрахунків, яка розроблялася Майклом Портером в 1980-х роках [1].

API (англ. Application Programming Interface) — прикладний програмний інтерфейс.

GNU GPL (англ. GNU General Public License) — одна з найпопулярніших ліцензій на вільне програмне забезпечення.

VM (англ. Virtual Machine) — віртуальна машина.

ОС — операційна система.

CI/CD (англ. ontinuous integration and continuous delivery) — підхід до автоматизації розробки програмного забезпечення.

ВСТУП

Сьогодні, чи не кожна галузь людської діяльності використовує для своїх цілей різноманітні програмні системи. Гідроакустика не є винятком. Основне завдання гідроакустики — дослідження поведінки (генерації, розповсюдження та затухання) звуку в воді. Практичне застосування гідроакустика знаходить у безлічі сфер, пов'язаних з водним середовищем, наприклад, підводна локація, виявлення об'єктів та аналіз їх розмірів, поведінки під водою як у цивільній так і у військовій сфері, дослідження природи морського чи океанського дна, тощо.

Відокремлення гідроакустики в окремий розділ акустичних наук відбулось зокрема через те, що поведінка звуку в водному середовищі та повітрі суттєво відрізняється. Звукові хвилі у воді можуть розповсюджуватись на значні відстані з малою втратою передачі. Особливий поштовх для розвитку гідроакустика отримала в другій половині XX-століття під час активного розвитку обчислювальних машин та періоду Холодної війни.

Для прогнозування затухань звукових хвиль в гідроакустиці використовуються різноманітні методи, зокрема можна виділити наступні:

— метод нормальних мод (нормальних амплітуд);

— використання швидкого перетворення Фур'є;

— променевий метод.

Одним з програмних засобів, що використовує метод нормальних мод, є програма KRAKEN, яка розроблялася Майклом Портером в 1980-х роках [1]. Розрахункові алгоритми були розроблені мовою FORTRAN. Протягом свого існування, програма KRAKEN була розширена та покращена і завоювала авторитет серед користувачів. Проте, на сьогодні, при аналізі наявних рішень, не було виявлено модифікації програми KRAKEN, або ж програмного забезпечення на базі будь-яких інших моделей, які використовують метод нормальних мод, що працює у вигляді веб-застосунку та не потребує встановлення додаткового програмного забезпечення для повноцінної роботи.

Головною метою роботи, яка виконується в рамках наукового напрямку лабораторії комп'ютерного моделювання динамічних процесів, є створення веб-сервісу для гідроакустичних розрахунків на основі моделей KRAKEN та популяризація методу нормальних мод. Розроблений веб-сервіс дозволить користувачам проводити гідроакустичні розрахунки, зокрема прогнозування затухань звукових хвиль, використовуючи метод нормальних мод, який описаний та реалізовний розробниками програми KRAKEN. Також веб-сервіс надаватиме додатковий функціонал, який спростить аналіз отриманих результатів.

Для розробки програмного забезпечення було використано середовище Microsoft Visual Studio мову програмування C# та технологію ASP.NET Core для розробки частини програми, що відповідає за розрахунки та мову програмуванні JavaScript разом з бібліотекою React.

1. ЗАДАЧА РОЗРОБКИ ВЕБ-СЕРВІСУ ГІДРОАКУСТИЧНИХ РОЗРАХУНКІВ ЗА МОДЕЛЯМИ СИСТЕМИ KRAKEN

Метою бакалаврської роботи є створення веб-сервісу гідроакустичних розрахунків на основі моделей KRAKEN. Тому реалізацію цього проекту можна умовно розбити на дві основні частини:

— аналіз методу нормальних мод та моделей програми KRAKEN;

— розробка веб-сервісу.

1.1. Аналіз методу нормальних мод та моделей програми KRAKEN

Метод нормальних мод вже багато років використовується у гідроакустиці. Одним з перших робіт по цій темі була опублікована у 1948 році ізраїльсько-американським вченим Пекерісом Х. Л., який розробив теорію для простої двошарової моделі, де розповсюдження звуку в кожному шарі відбувається зі константною швидкістю [2]. Суттєвий поштовх в розвитку методів, що ґрунтуються на нормальних модах, зробив своїми працями американський вчений Вільямс А. О. [3].

На основі цих праць, та з метою покращення точності вже розроблених моделей і почалась робота над програмою KRAKEN. Розв’язок на основі нормальних мод передбачає вирішення одновимірного рівняння, яке дуже схоже на рівняння, яке вирішується при дослідженні задач на коливання струни [1]. У своїй праці, автор програми KRAKEN Портер М. Б. детально описує метод нормальних мод, починаючи з найбільш грубих методів підрахунку та поступово вдосконалюючи їх для того щоб можна було моделювати проблеми, які бувають при розв’язанні задач з реального світу, наприклад здатність обробляти багатошарові середовища, врахування різних граничних умов тощо.

Модулі програми KRAKEN групуються у так звані моделі, які не мають жорсткої залежності одна від одної. Підрахунок та аналіз зменшення інтенсивності звуку за допомогою програми відбувається в три етапи з використанням наступних компонентів:

— моделі типу KRAKEN для обчислення нормальних мод;

— моделі типу FIELD, щоб підрахувати тиск звукових хвиль на заданих глибинах;

— функції для побудови графіків (PLOTFIELD, PLOTMODE, PLOTTLR).

1.2. Розробка веб-сервісу

Розроблений веб-сервіс повинен не просто надавати графічний інтерфейс до програми KRAKEN, а використовувати власні функції для підрахунків. Це дозволить надалі підтримувати та оновлювати програму, додаючи новий або змінюючи наявний функціонал.

До функціональних можливостей розроблюваного веб-застосунку було висунуто наступні вимоги:

— обчислення величин за моделями системи KRAKEN;

— висока точність отриманих значень;

— зручний, інтуїтивно зрозумілий графічний інтерфейс користувача;

— взаємодія з користувачем через графічний інтерфейс (виведення помилок, попереджень);

— перевірка вхідних даних на стороні клієнта та сервера;

— можливість завантаження вхідних даних з файлу;

— можливість збереження введених даних для повторного використання;

— експорт табличних даних у наступних форматах: *.xls, *.csv, XML, JSON та експорт графіків у вигляді зображень;

— наявність допоміжної документації для користувача.

2. ОПИС ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ

2.1. KRAKEN

KRAKEN — оригінальна програма для гідроакустичних розрахунків на основі нормальних мод. Останнім часом, коди моделей програми KRAKEN повторюються в пакеті моделювання Acoustic Toolbox [4], який розповсюджується у вільному доступі під ліцензією GNU GPL та з відкритим вихідним кодом. Усього до пакета Acoustic Toolbox входять чотири моделі, для акустичних розрахунків: BELLHOP, KRAKEN, SCOOTER, SPARC. Кожна модель реалізована мовою програмування Fortran, та у вигляді функцій для MATLAB. При розробці програмного продукту для бакалаврської роботи було вирішено використовувати саме коди програми написані Fortran для аналізу та реалізації алгоритмів та порівняння кінцевих результатів програми.

2.1.1. Реалізація KRAKEN для Fortran

Fortran — це мова програмування, яка добре підходить для інтенсивних та наукових обчислень. Розроблена у середині минулого століття корпорацією IBM, вона ще досі використовується для високопродуктивних обчислень. Сучасні коди програми KRAKEN відповідають стандарту Fortran90.

Для використання програми користувачу, у багатьох випадках потрібно буде самому скомпілювати програму, тому що при компіляції враховуються особливості операційної системи. Моделі типу KRAKEN та FIELD компілюються окремо, що додає певної гнучкості при використанні. Скомпільована програма — це виконуваний файл, який користувач може викликати з командного рядка, передаючи аргументи. Вхідні дані для програми подаються у вигляді файлів, під час роботи програма генерує текстові файли, призначені для аналізу даних користувачем та бінарні файли, які використовуються іншими моделями чи

функціями. Схема взаємодії між компонентами програми KRAKEN та зовнішніми файлами зображена на рисунку 2.1.

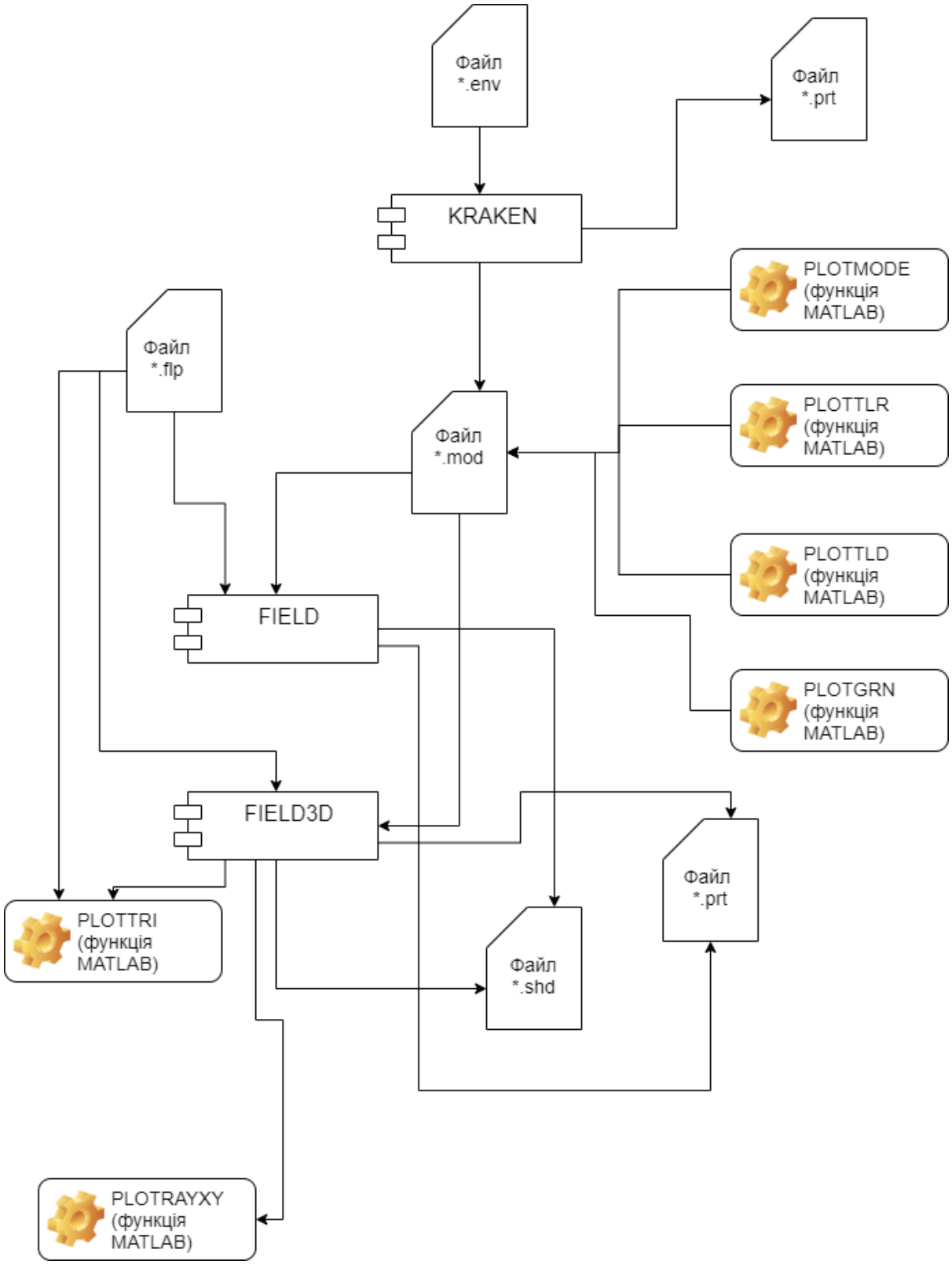


Рисунок 2.1 — Схема взаємодії між компонетами програми KRAKEN та зовнішніми файлами

З рисунка 2.1 видно, що сама тільки реалізація програми мовою Fortran, не надає користувачу можливість повністю проаналізувати результати обчислень, для цього потрібно встановлювати додаткове програмне забезпечення, а саме MATLAB. Також не має змоги без змін у програмному коду чи без використання додаткових утиліт переглянути підраховані нормальні моди та тиск акустичного поля, тому що вони записуються у бінарні файли.

2.1.2. Реалізація KRAKEN для MATLAB

Реалізація на MATLAB дозволяє користувачу проводити аналогічні гідроакустичні розрахунки з додатковим функціоналом для перегляду результатів. Користувачу не надається графічний інтерфейс, отримання вхідних даних та виведення результатів відбувається також з використанням файлів. Дана версія надає дещо кращий інструментарій для аналізу результатів обчислень, ніж Fortran версія KRAKEN. Зокрема користувач без додаткових засобів може здійснювати побудову графіків, наприклад відобразити нормальні моди та втрату передачі звуку (рисунок 2.2 та рисунок 2.3 відповідно).

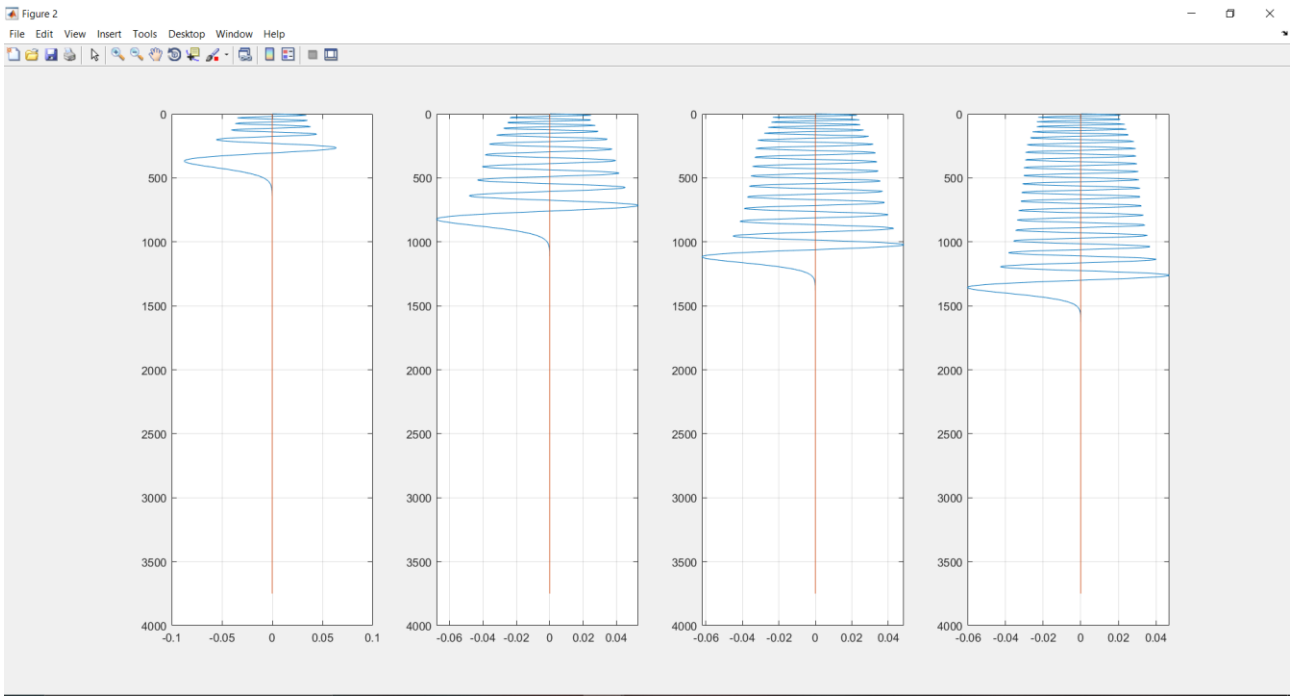


Рисунок 2.2 — Відображення нормальних мод на графіку використовуючи функції для MATLAB

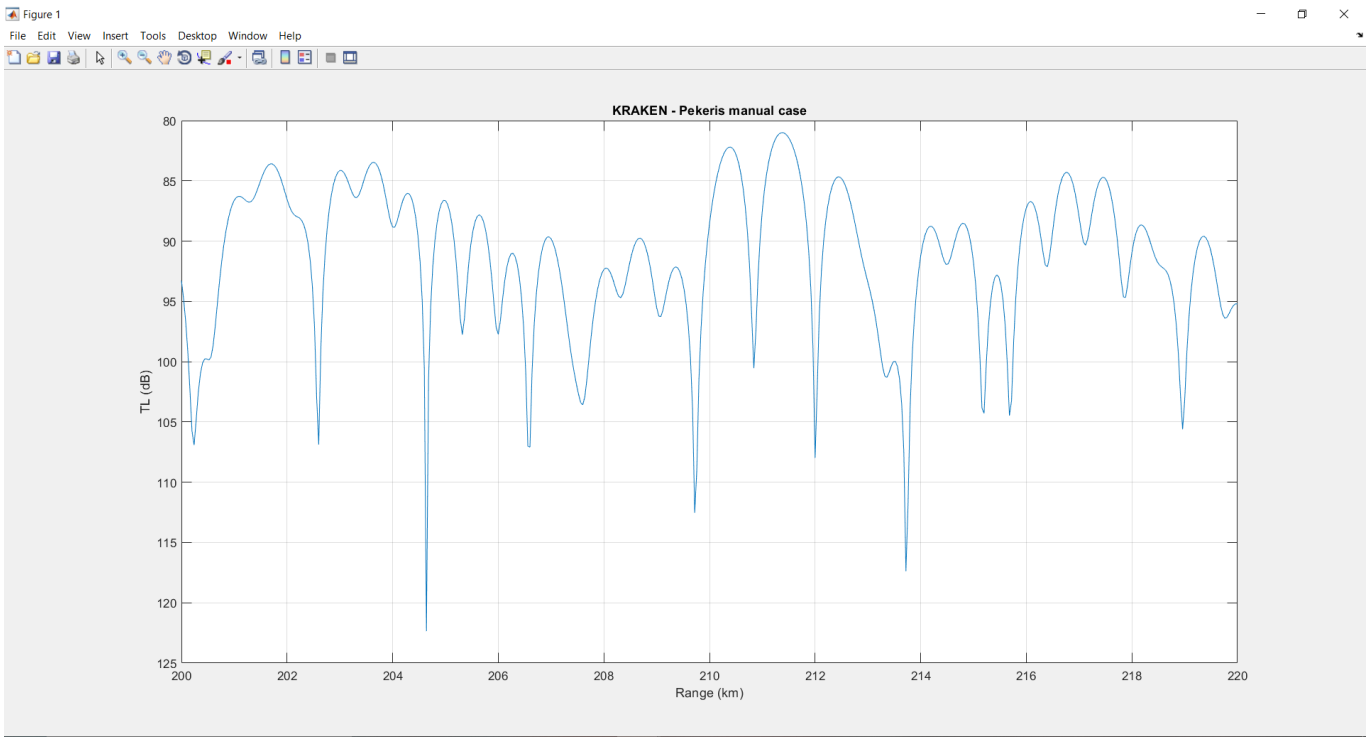


Рисунок 2.3 — Відображення втрати передачі звуку на графіку використовуючи функції для MATLAB

Також, використовуючи KRAKEN для MATLAB, користувач може записувати результат виконання в змінні і виводити їх значення в потрібній для себе формі. До недоліків можна віднести відсутність графічного інтерфейсу.

2.2. MATLAB KRAKEN

Модифікація MATLAB KRAKEN, розроблена Б.Душавом з Вашингтонського університету — це перероблена версія KRAKEN з MEX файлом, яку можна викликати з MATLAB [5]. MEX файл – це тип комп'ютерного файлу, який забезпечує зв'язок між MATLAB та кодом, що написаний мовами C, C++, Fortran. В даній версії, розрахункові алгоритми реалізовані мовою Fortran.

Ця версія програми KRAKEN не проводить розрахунків тиску звукового поля, тобто тих за які відповідає модель FIELD, а тільки рахує нормальні моди. Вхідні дані вона отримує з файлу MATLAB у вигляді параметрів функції. Результат

виконання записується у змінні, що дає змогу аналізувати результати обчислень, зокрема у вигляді графіків, як показано на рисунку 2.4.

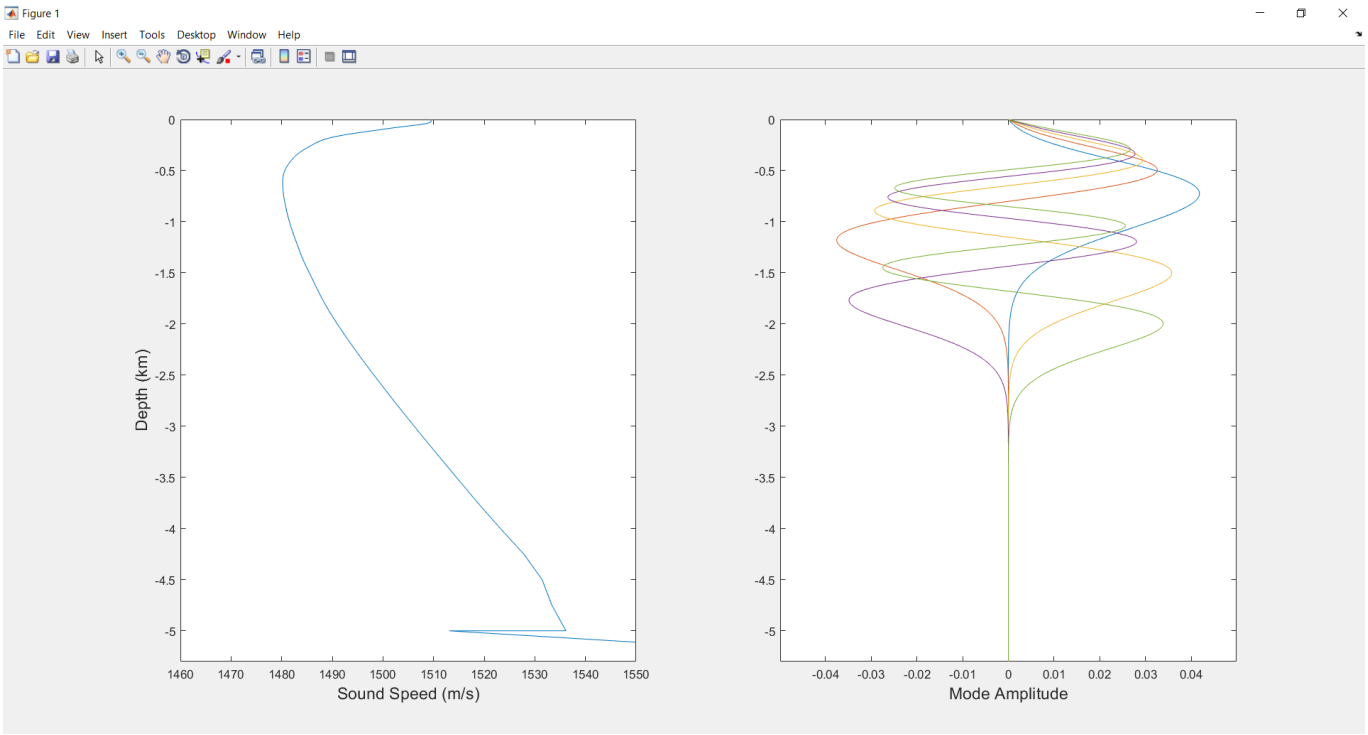


Рисунок 2.4 — Графіки на основі результатів обчислень MATLAB KRAKEN

Серед переваг цього програмного продукту можна виділити те, що він надає можливості для аналізу результату, які доступі в MATLAB, і проводить обчислення зі швидкістю програм написаних мовою Fortran. Недолік — відсутність можливості проводити обчислення тиску акустичного поля, а отже й втрату передачі звуку.

2.3. KRAKEN for Python

Розроблена О.Родрізесом з Алгарвського університету, KRAKEN for Python надає функціонал, написаний мовою Python, для запису та читання файлів, які використовує оригінальна програма KRAKEN [5]. Користувач задає вхідні дані, присвоюючи значення змінним на мові Python, потім програма створює текстові файли, необхідні для роботи KRAKEN, та викликає виконуваний файл. В Kraken for Python також реалізовані методи для читання зі згенерованих бінарних файлів та

запис отриманих значень в змінні, що дає змогу будувати таблиці та графіки (рисунок 2.5).

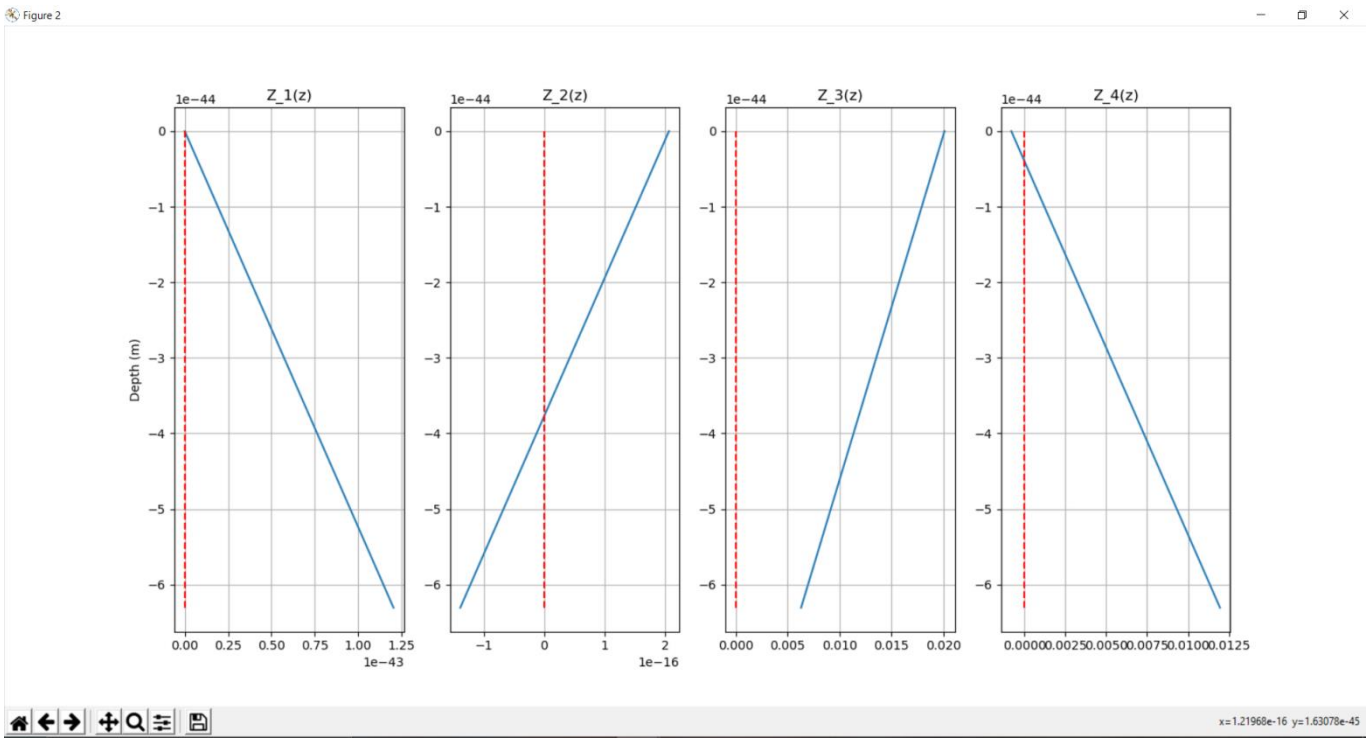


Рисунок 2.5 — Графічне відображення результатів засобами Kraken for Python

Тобто, Kraken for Python — це легка та продуктивна обгортка над програмою KRAKEN для задання вхідних даних та читання результатів. Завдяки цій модифікації, користувачу не потрібно використовувати засоби MATLAB для графічного аналізу результату, зокрема у вигляді графіків.

3. ЗАСОБИ РОЗРОБКИ

3.1. Середовище розробки Microsoft Visual Studio

Для розробки серверної та клієнтської частини системи використовувалось Microsoft Visual Studio 2019, інтерфейс якого зображений на рисунку 3.1.

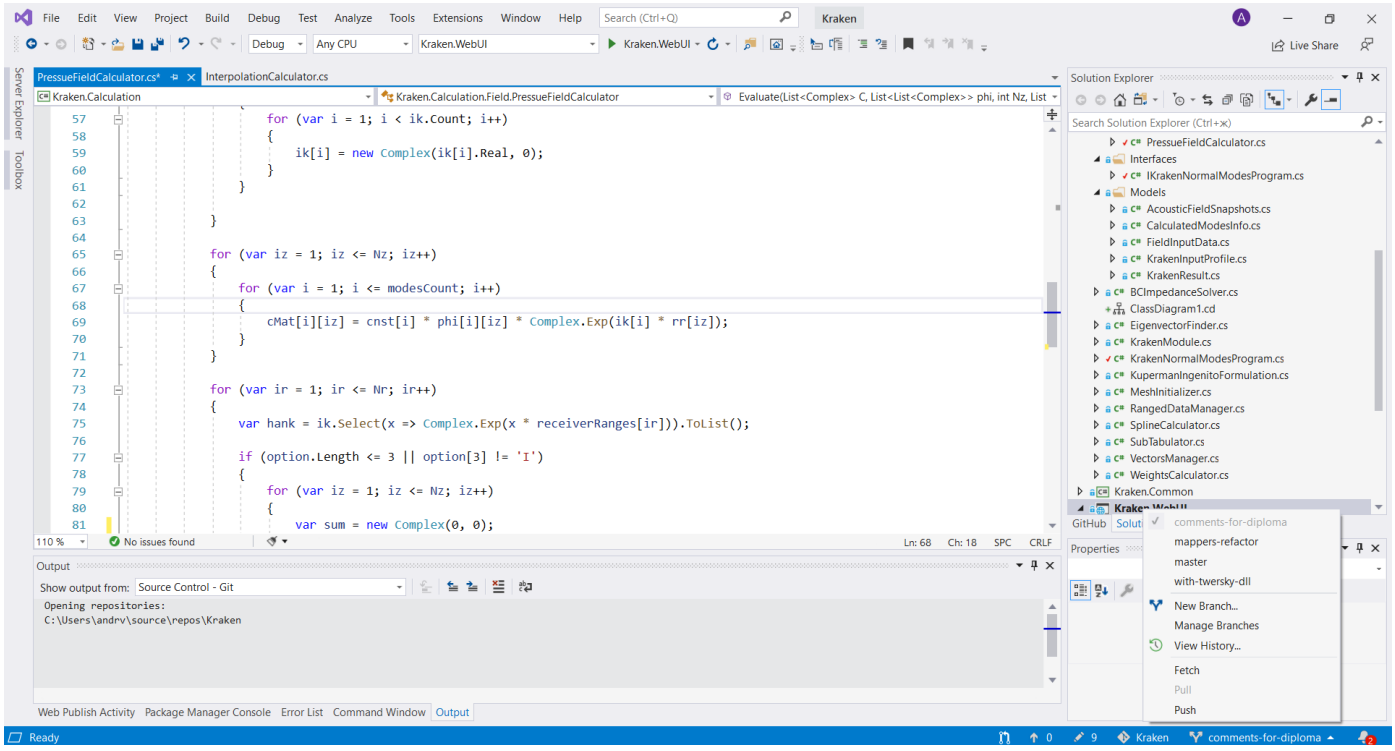


Рисунок 3.1 — Середовище розробки Microsoft Visual Studio 2019

Microsoft Visual Studio — це інтегроване середовище розробки від компанії Microsoft [6]. Воно може використовуватись для розробки найрізноманітніших типів застосунків: консольні застосунки, застосунки для комп'ютерів використовуючи платформу Windows Forms та Windows Presentation Foundation, веб-застосунків, мобільні застосунки тощо. Використовуючи інтерфейс Microsoft Visual Studio, можна проводити інтеграцію з хмарними технологіями та адмініструвати сервер баз даних.

Середовище розробки Microsoft Visual Studio дозволяє проводити налагодження кодів програм, написаних різними мовами та виконувати їх власне з інтерфейсу середовища.

Також обране середовище розробки надає вбудовану підтримку розподіленої системи керування версіями файлів Git. Одразу з інтерфейсу Microsoft Visual Studio користувач має змогу працювати як з локальними так і з віддаленими репозиторіями.

Microsoft Visual Studio дозволяє встановлювати засоби для підтримки мов програмування чи технологій за потреби, що зменшує кількість необхідних ресурсів, потрібних для роботи.

3.1. Мова програмування C# та платформа .NET Core

Для реалізації основних алгоритмів розрахунку та логіки системи було обрано мову програмуваня C#. Мова C# – сучасна мова програмуваннння об’єктно-орієнтованого спрямування з безпечною типізацією. Перша версія мови вийшла разом з релізом Microsoft Visual Studio .NET в лютому 2002 року. Поточною версією мови є версія C # 8.0, яка вийшла в 2 квітня 2019 року разом з Visual Studio 2019 [7]. На ній пишуться найрізноманітніші програми: від невеликих десктопних програм до високо навантажених веб-порталів і веб-сервісів, які обслуговують щодня мільйони користувачів. Тому, останні декілька років, мова C# завжди знаходиться на верхівці рейтингу мов програмування та поступово покращує свої позиції. Вона відноситься до широкого кола C-подібних мов програмування.

C# підлаштовується під вимоги ринку, та з кожним релізом додає новий функціонал. Надаючи можливості функціонального та подійно-орієнтованого програмування — мова C# дещо відходить від об’єктно-орієнтованої парадигми, яка є базою мови, проте таким чином розробники отримують більше гнучкості для побудови сучасних систем.

Мова C# нерозривно пов’язана з платформою .NET, зокрема з найсучаснішою версією .NET Core. .NET Core — платформа для розгортання та розробки

програмних продуктів, з відкритим вихідним кодом, що розробляється та підтримується компанією «Microsoft». Вона є прямим спадкоємцем платформи .NET, але на відміну від попередника, є кросплатформною, тобто може працювати на більшості сучасних операційних систем, таких як: Windows, Linux та MacOS. .NET Core дає можливість використовувати в одному програмному продукті три мови програмування: Visual Basic, F# та C#. Тобто, C# — це лише інструмент для написання коду для платформи .NET Core.

Сама платформа .NET Core складається з двох частин: набору інструментів та бібліотек для розробників — CoreFX і CoreCLR — середовища для виконання скомпільованого .NET коду [8]. CoreFX надає значну кількість готових бібліотек для роботи з колекціями, файлами, мережевими протоколами тощо, що розповсюджуються у вигляді незалежних пакетів. Завдяки відкритому вихідному коду, ці бібліотеки активно розвиваються і підтримуються спільнотою розробників. CoreCLR в свою чергу є головною частиною платформи .NET Core, тому що вона забезпечує виконання коду, написаного на мовах програмування .NET на різних операційних система з різною архітектурою процесора. Подібно до віртуальної машини для мови Java, завдяки CoreCLR розробники не можуть просто так отримати можливість прямого читання з оперативної пам’яті або ж прямої взаємодії з операційною системою. CoreCLR також відповідальна за механізм збирання сміття, вивільняючи пам’ять від ресурсів, які програма не буде використовувати в надалі при своїй роботі.

3.2. Фреймворк ASP.NET Core

ASP.NET Core — безкоштовний фреймворк з відкритим вихідним кодом, створений компанією Microsoft, призначений для створення різних веб-додатків: від невеликих веб-сайтів до веб-порталів і веб-сервісів, які обслуговують щодня мільйони користувачів [9]. Автори фреймворку рекомендують використовувати його разом з платформою .NET Core, проте ASP.NET Core також може працювати зі старішою версією .NET Framework.

Фреймворк побудований за принципом модульності, розробники фреймворку вирішили створити максимально легку готову до роботи версію фреймворку, а розробники, які будуть використовувати його, зможуть за потреби встановлювати потрібні бібліотеки та модулі, використовуючи, наприклад, пакетний менеджер NuGet.

ASP.NET Core дозволяє створювати веб-додатки, які генерують представлення в браузер на стороні сервера і водночас підтримує вбудовану інтеграцію з технологіями розробки клієнтського інтерфейсу, такими як: Angular, React.js, Vue.js.

Завдяки тому, що додатки, написані на ASP.NET Core можуть виконуватись на різних операційних системах, розробки отримують підтримку контейнеризації – можливість запускати додаток в контейнерах Docker, та системах оркестрації контейнерами Kubernetes, додає ще більше гнучкості при розробці застосунків та зменшує вартість розгортання програмного забезпечення.

3.3. React

React – це бібліотека мови JavaScript з відкритим вихідним кодом для побудови графічного інтерфейсу користувача [10]. Розробники React популяризують підхід до розробки веб-інтерфейсу, при якому для побудови інтерфейсу використовують багаторазові компоненти, які відображають данні, що змінюються протягом часу. Бібліотеку React можна використовувати для побудови односторінкових застосунків.

Фундаментальним поняттям для бібліотеки React є virtual DOM. DOM (Document Object Model) — надає можливість представлення структури документа за допомогою об'єктів. Це кросплатформний засіб, який не залежить від мови програмування, для представлення та взаємодії з даними у HTML, XML. Веб-браузери підтримують складові DOM, і ми можемо взаємодіяти з ними, використовуючи JavaScript та CSS. Ми можемо працювати з вузлами документа, змінювати дані, знімати та вставляти нові вузли. Проблема DOM в тому, що він не

був розроблений для побудови динамічних веб-сторінок, де дані змінюються незалежно в різних місцях.

Virtual DOM був створений для того, щоб замість взаємодії з DOM напряму, розробники працювати з його полегшеною копією [11]. Можна вносити зміни у віртуальну копію, а після цього застосовувати зміни до реального DOM. При цьому відбувається порівняння DOM-дерева з його віртуальною копією, визначається різниця і запускається рендеринг (зміна) тільки тих об’єктів, які піддались зміні. Такий підхід працює швидше, бо не включає в себе всі ресурсозатрачувані операції реального DOM.

React оновлює реальний DOM в три етапи (рисунок 3.2):

— зміни стану компонента у virtual DOM;

— розрахунок різниці між старим та оновленим virtual DOM;

— оновлення реального DOM якщо між версіями virtual DOM є різниця.

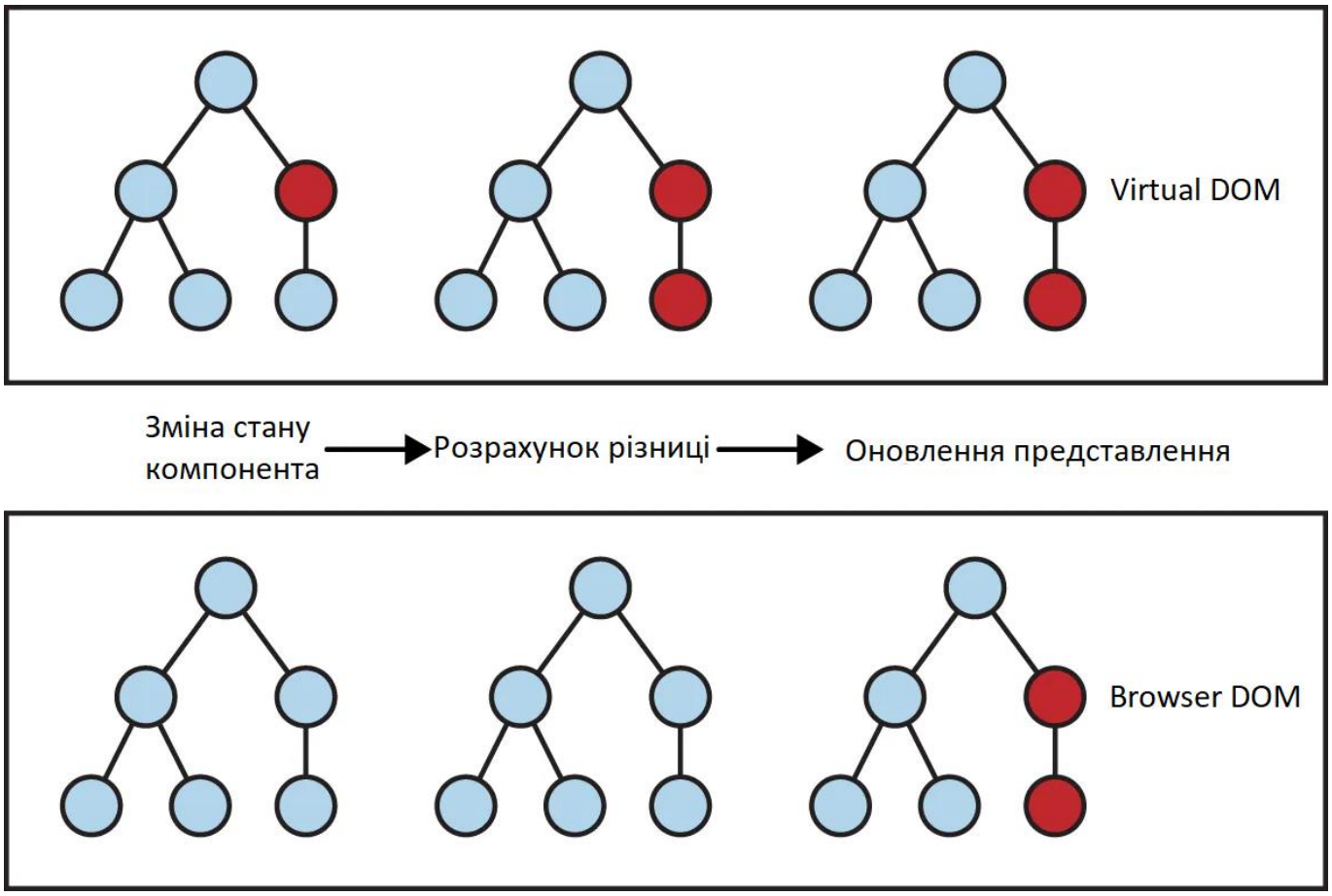


Рисунок 3.2 — Оновлення веб-сторінки з використанням virtual DOM

Однією з особливостей, що відрізняє React від інших засобів для розробки клієнтського інтерфейсу є можливість застосування мови JSX. JSX — синтаксис, схожий на XML, близький до HTML, але не зовсім HTML. Це фактично JavaScript з вставками HTML [12]. За допомогою мови JSX, можна поєднати логіку рендерингу та HTML розмітку компонента, а не виносити їх в окремі файли.

3.4. Платформа контейнеризації Docker

Життєвий цикл програмного продукту також включає етап розгортання, тобто дії, які роблять систему готовою до використання. В дипломному проекті, для полегшення етапу розгортання веб-сервісу використовується Docker. Docker — інструмент з відкритим вихідним кодом, який автоматизує розгортання застосунку у середовищах, що підтримують контейнеризацію [13]. Під час розробки застосунку потрібно під’єднати всі необхідні залежності: на зразок бібліотек, веб-сервера, баз даних тощо. Інакше застосунок може працювати на комп’ютері розробника, але не запуститися на зовнішньому сервері чи комп’ютерах інших розробників та тестувальників. Цю проблему можна розв’язати, ізолювавши застосунок від системи.

Раніше для розгортання застосунків часто використовували віртуальну машину. Проблема у тому, що VM — це додаткова операційна система, яка запускається поверх хостової, і це суттєво збільшує витрати на оплату серверу. Docker ж розділяє ядро ОС між усіма застосунками, що виконуються — контейнерами, що працюють як окремі процеси хостової ОС [14]. Різниця між розгортанням застосунків на віртуальній машині та в якості Docker контейнера зображена на рисунку 3.3.

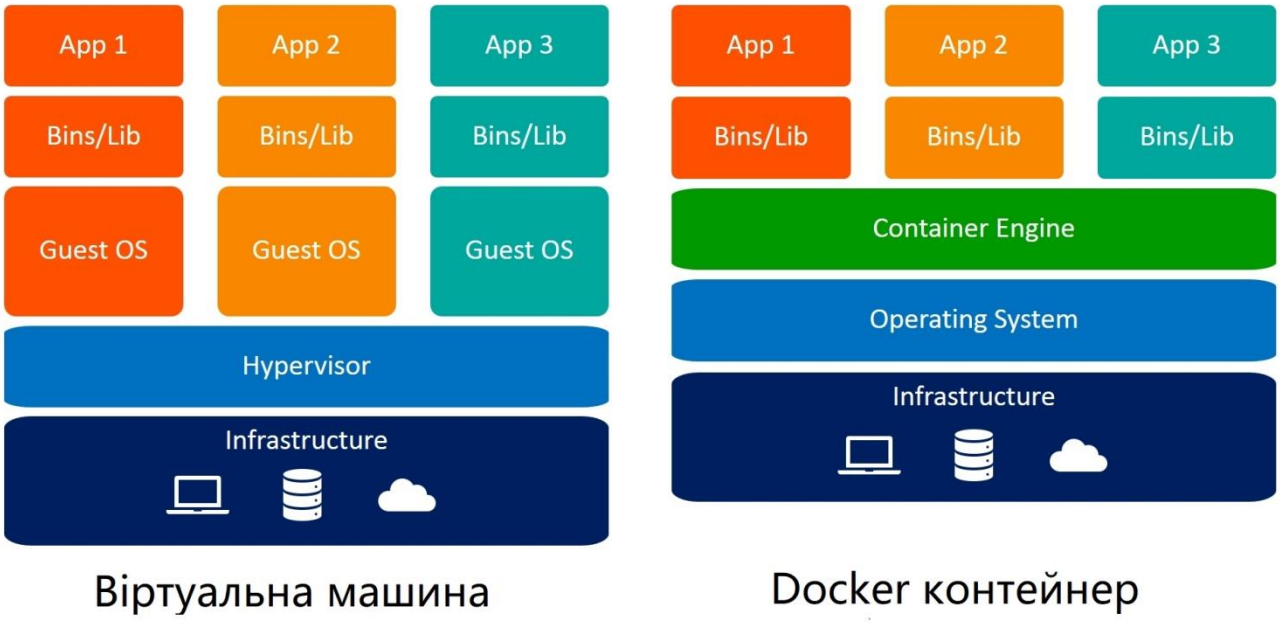


Рисунок 3.3 — Принцип роботи віртуальної машини та Docker контейнера

Серед переваг Docker над віртуальною можна виділити наступні:

— зменшення ваги застосунку;

— зручна інкапсуляція застосунку;

— збільшення швидкості запуску;

— зручність масштабування.

3.5. Microsoft Azure

Для розгортання та хостингу веб-сервісу використовувались засоби Microsoft Azure. Microsoft Azure — це платформа для хмарних обчислень, створена компанією Microsoft. Вона надає широкий спектр хмарних сервісів для обчислень, аналітики, зберігання даних, організації мережі. Сервіс надає численну кількість пропозицій, тому користувач платить тільки за ті ресурси для можливості, які використовує.

Власне для розгортання і хостингу розробленого веб-сервісу використовувалася служба Azure App Service. Azure App Service — це сервіс, що базується основі HTTP, для розміщення веб-додатків, REST API, та серверних частин для мобільних застосунків. Сервіс може працювати з додатками,

розробленими основними мовами програмування. Застосунки можуть виконуватись в середовищі Linux та Windows.

App Service дозволяє підвищити безпеку застосунку, аналізувати завантаженість (рисунок 3.4), проводити автоматичне масштабування та автоматизоване управління. Також AppService надає можливості для CI/CD, для безперервного розгортання з GitHub, Docker Hub чи інших сервісів, які надають можливість розміщення вихідних кодів застосунків.

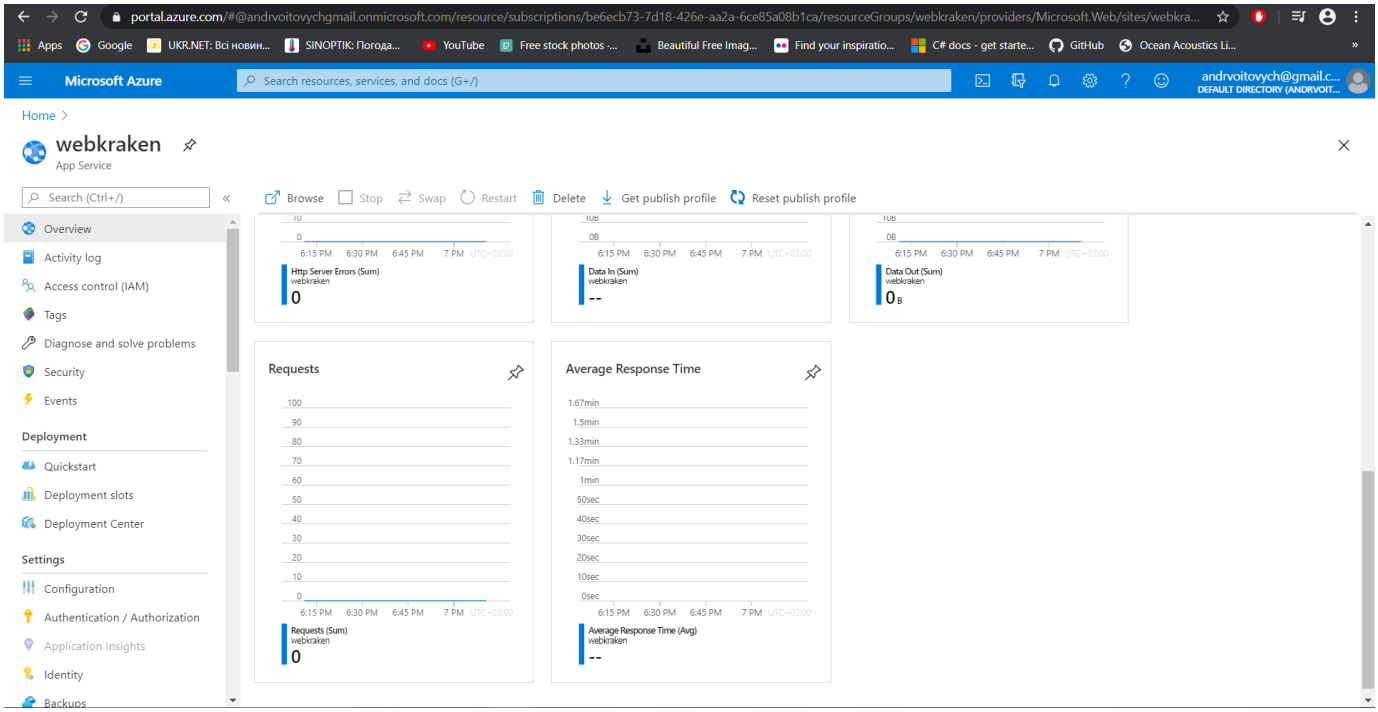


Рисунок 3.4 — Інтерфейс Azure App Service для аналізу завантаженості веб-застосунку

Також однією з причин, через які для розміщення веб-сервісу було обрано служби Microsoft Azure — це полегшена інтеграція з іншими продуктами, що належать компанії Microsoft. Наприклад, для публікації та розгортання розробленого застосунку в Azure App Service, використовувалось середовище розробки Microsoft Visual Studio.

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

4.1. Архітектура системи

Процес розробки будь-якого успішного програмного забезпечення починається з аналізу та специфікації вимог, за якими йде проектування, і тільки проаналізувавши та зрозумівши систему можна приступати до етапу реалізації (кодування).

4.1.1. Архітектура клієнт-сервер

Проаналізувавши вимоги до розроблюваного програмного продукту, було вирішено використовувати архітектуру клієнт-сервер. Архітектура клієнт-сервер є одним із найрозповсюдженіших принципів побудови архітектури веб-орієнтованого програмного забезпечення. Головна концепція цього типу архітектури полягає в тому, що в клієнт-серверних інформаційних мережах основна частина даних та ресурсів знаходиться в серверах, які обслуговують клієнтів [15].

У розробленій системі роль клієнта виконує додаток з графічним інтерфейсом, розроблений на React, а роль сервера — застосунок ASP.NET Core WebApi (рисунок 4.1).

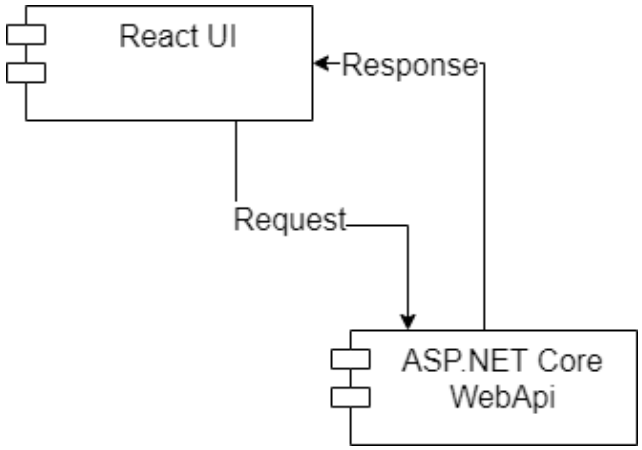


Рисунок 4.1 — Клієнт та сервер розробленої системи

Клієнт робить запити на сервер використовуючи протокол HTTP, а сервер повертає необхідні дані. Завдяки тому, що система побудована за такою архітектурою, сервер не знає про існування клієнту, він просто робить необхідні обчислення. Це надає змогу додавати нових клієнтів без зміни коду сервера, наприклад мобільного додатку, іншого веб-інтерфейсу чи десктопної програми.

4.1.2. Структура проекту

Структура розробленого програмного проекту (рисунок 4.2), завдяки групуванню логічно пов’язаних класів та методів в окремі модулі, які пов’язані з іншими частинами системи за допомогою інтерфейсів з використанням принципу інверсії залежностей, що сприяє розширенню системи та написання коду, який піддається тестуванню [16].

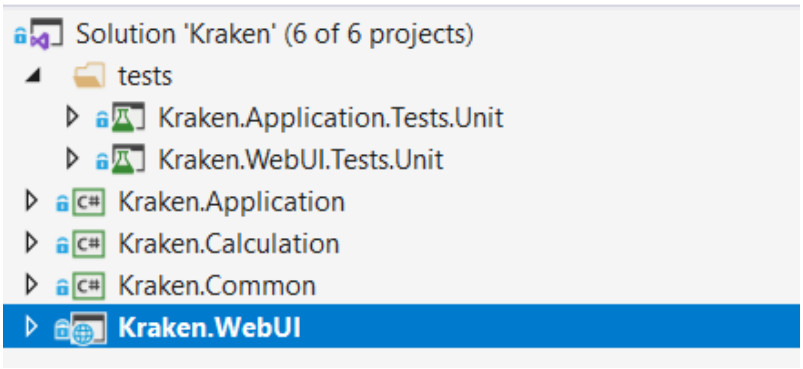


Рисунок 4.2 — Структура проекту

Структура проекту включає наступні компоненти:

— бібліотека класів Kraken.Calculation об’єднує класи та інтерфейси, які реалізують обчислення на основі моделей KRAKEN. Не має залежностей від інших компонентів системи, тому може розповсюджуватись незалежно і за потреби використовуватись в інших системах;

— бібліотека класів Kraken.Common об’єднує інтерфейси та класи, які не містять власне бізнес логіки додатку та можуть використовуватись в різних частинах проекту;

— бібліотека класів Kraken.Application об’єднує інтерфейси та класи, які є бізнес-логікою додатку. Використовує класи з бібліотеки Kraken.Calculation для проведення обчислень;

— проект типу ASP.NET Core Web Application Kraken.WebUI є власне серверною частиною системи. Відповідає за обробку вхідних даних від клієнта та передачі їх для обробки класам Kraken.Application. Також в цьому проекті відбувається впровадження залежностей;

— проект типу xUnit Test Project Kraken.Application.Tests містить програмні тести компонентів проекту Kraken.Application;

— проект типу xUnit Test Project Kraken.WebUI.Tests містить програмні тести компонентів проекту Kraken.WebUI.

4.2. Реалізація моделей системи KRAKEN

Реалізації моделей системи KRAKEN мовою C# відбувалась, ґрунтуючись на коди оригінальної програми KRAKEN, написаної мовою програмування Fortran. Програма KRAKEN розроблялася в часи, коли парадигма об’єктно-орієнтованого програмування тільки зароджувалась і використання операторів безумовного переходу “goto” було нормою, тому коди програми не завжди структуровані та зрозумілі. Також складності в реалізації додавало те, що індексація масивів в мові Fortran починається з одиниці, а не з нуля як у C# і більшості інших сучасних мов програмування.

Для реалізації моделей типу KRAKEN, які використовуються для підрахунку нормальних мод, було створено інтерфейс IKrakenNormalModesProgram та його реалізацію — клас KrakenNormalModesProgram (рисунок 4.3).

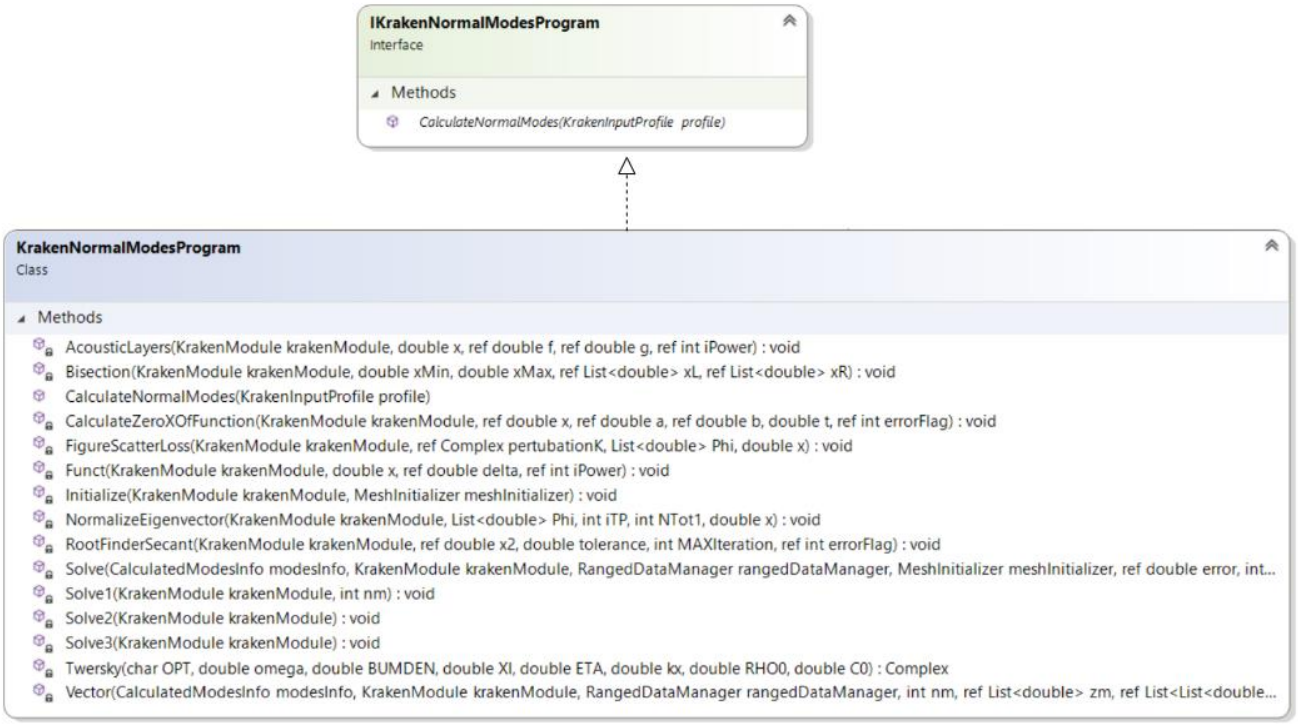


Рисунок 4.3 — Структура інтерфейсу IKrakenNormalModesProgram та класу KrakenNormalModes

В оригінальній програмі вхідні дані поступово зчитувались з файлу та використовувались для розрахунків. В розробленій програмі модуль, який відповідає за розрахунки не залежний від джерела вхідних даних та потребує для роботи об’єкт типу KrakenInputProfile, структура якого зображена на рисунку 4.4.

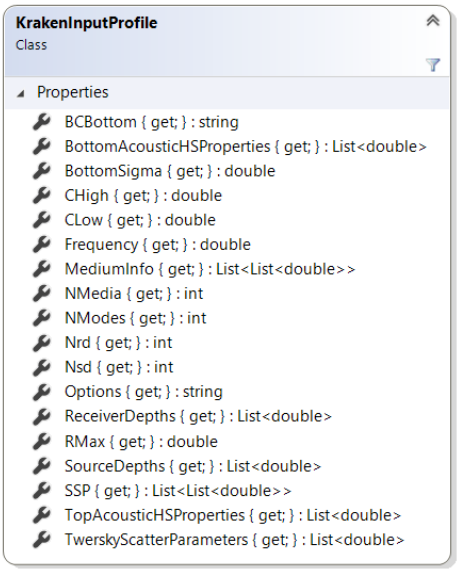


Рисунок 4.4 — Клас KrakenInputProfile, який описує структуру вхідних даних

Результатом успішних обчислень є об’єкти типу KrakenResult та CalculatedModesInfo (рисунок 4.5).

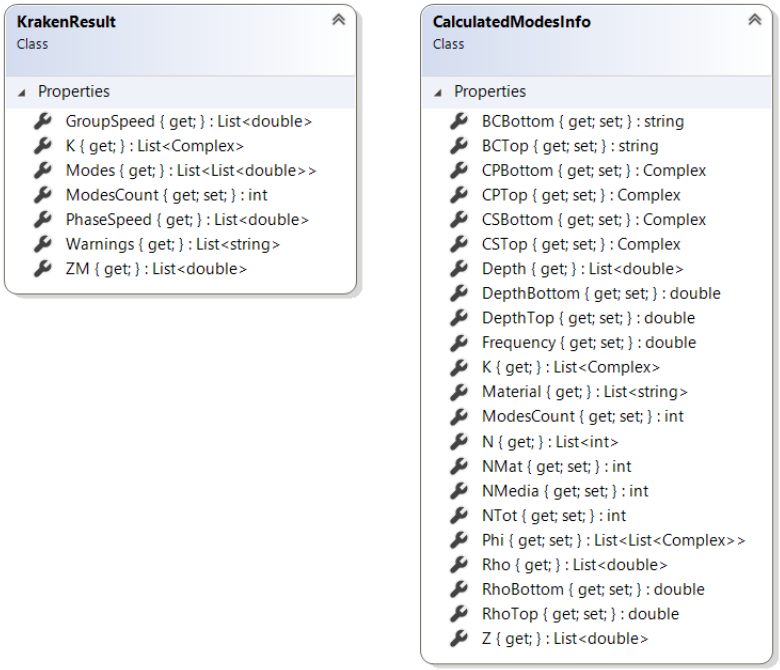


Рисунок 4.5 — Класи KrakenResult та CalculatedModesInfo

Клас KrakenResult міcть в собі наступні підраховані величини: групові швидкості, хвильові числа, фазові швидкості для кожної зі знайдених нормальних мод, кількість нормальних мод, нормальні моди, глибини, на яких обчислювались моди та список повідомлень про можливі неточності, що виникли при обчисленнях. Клас CalculatedModesInfo описує структуру даних, необхідну для роботи моделі FIELD і є аналогом файлу з розширенням *.mod, який генерує при роботі оригінальна програма KRAKEN.

Розрахунок нормальних мод та тисків акустичного поля може відбуватись назалежно. Алгоритми моделі FIELD реалізовані окремо та представлені інтерфейсом IFieldProgram та його реалізацією FieldProgram (рисунок 4.6).

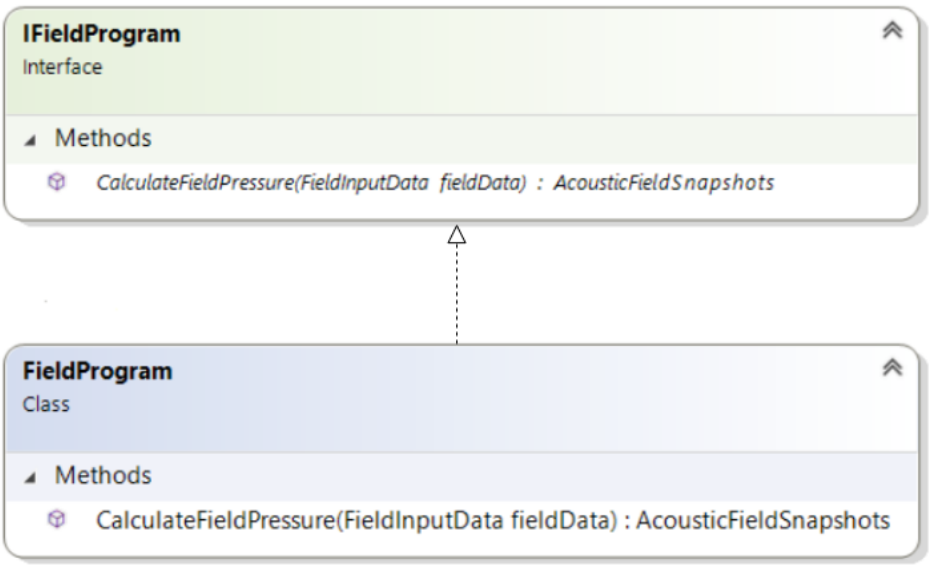


Рисунок 4.6 — Структура інтерфейсу IFieldProgram та класу FieldProgram

Вхідні дані для обчислень інкапсулюються в екземплярі класу FieldInputData, структура якого зображена на рисунку 4.7.

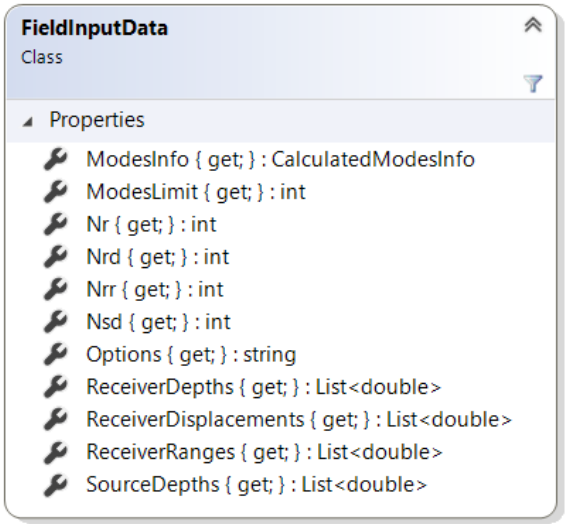


Рисунок 4.7 — Клас KrakenInputProfile, який описує структуру вхідних даних

Результатом успішних обчислень є об’єкт класу AcousticFieldSnapshots (рисунок 4.8), який містить підрахований тиск акустичного поля та величини, які необхідні для побудови графіків та таблиць, такі як: глибини джерела й приймача, діапазони приймача; та повідомлення про можливі неточності в обчисленнях.

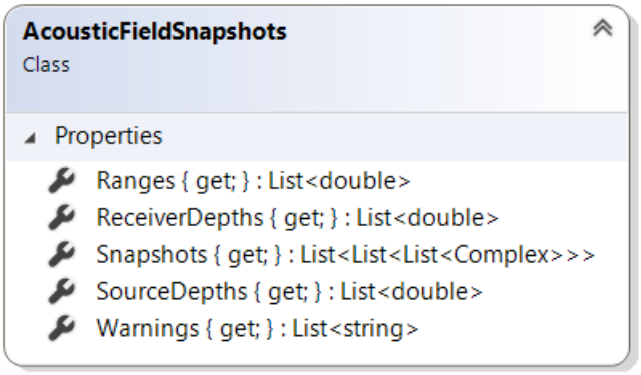


Рисунок 4.8 — Клас AcousticFieldSnapshots

Вищеописані класи для проведення обчислень також використовують допоміжні класи, які використовуються наприклад для пошуку квадратного кореня, інтерполяції, табуляції діапазону значень тощо.

Розроблені класи дозволяють проводити гідроакустичні розрахунки на основі моделей KRAKEN та можуть поширюватись у вигляді бібліотеки класів, яку можна підключати та використовувати в різних проектах. Бібліотека не має зовнішніх залежностей і використовує тільки вдубовані компоненти .NET Standart 2.0.

4.3. Обробка вхідних даних

У розробленому веб-сервісі, дані між клієнтом та сервером передаються у форматі JSON. У цьому форматі дані представляються у текстовому вигляді та можуть, без додаткових перетворень, бути прочитані людиною. Формат JSON підтримує примітивні типи даних — String, Number, Boolean, Null та комплексні — Object, Array, що дає змогу описувати складні об’єкти. Формат був розроблений Дугласом Крокфордом для передачі інформації мережею [17], проте зараз він також широко використовується для зберігання конфігураційних даних. Однією з додаткових переваг формату JSON є те, що його підтримка є вбудованою в мові програмування JavaScript, на якій і написаний клієнтський додаток.

За допомогою формату JSON у системі вдалось описати вхідні дані, які в програмі KRAKEN записувались у текстові файлі з розширенням *.env та *.flp, приклад об’єкту з вхідними даними зображено на рисунку 4.9.


```json
{
  "frequency": 10,
  "nModes": 44,
  "nMedia": 1,
  "topBCType": "V",
  "interpolationType": "N",
  "attenuationUnits": "F",
  "addedVolumeAttenuation": "",
  "zt": 0,
  "cpt": 0,
  "cst": 0,
  "rhot": 0,
  "apt": 0,
  "ast": 0,
  "bumDen": 0,
  "eta": 0,
  "xi": 0,
  "mediumInfo": [[500,0,5000]],
  "ssp": [[0,1500,0,1,0,0],[5000,1500,0,1,0,0]],
  "bottomBCType": "A",
  "sigma": 0,
  "zb": 5000,
  "cpb": 2000,
  "csb": 0,
  "rhob": 2,
  "apb": 0,
  "asb": 0,
  "cLow": 1400,
  "cHigh": 2000,
  "rMax": 1000,
  "nsd": 1,
  "sd": [500],
  "nrd": 1,
  "rd": [2500],
  "nModesForField": 9999,
  "nr": 501,
  "r": [200,220],
  "nsdField": 1,
  "sdField": [500],
  "nrdField": 1,
  "rdField": [2500],
  "nrr": 1,
  "rr": [0],
  "calculateTransmissionLoss": true,
  "sourceType": "R",
  "modesTheory": "A"
}
```


Рисунок 4.9 — Приклад вхідних даних системи

Одним з найголовніших процесів над даними в більшості систем є їх перевірка на коректність, тобто валідація. Некоректні дані, які не відповідають визначеним обмеженням можуть викликати збій в роботі програми. А поміщати власне в алгоритми розрахунків логіку для перевірки вхідних даних не є зразковою технікою проектування.

Серед негативних наслідків відсутності валідації можна виділити:

— неможливість відновити роботу після збою. Інколи програма може виконувати незворотні дії, наприклад відправлення даних мережею, які не можна скасувати;

— додаткове навантаження на систему. Відновлення роботи після збою і правила перевірки в логіці системи — це додаткове ускладнення алгоритмів роботи;

— складність ідентифікації проблеми. Якщо помилка виникла десь в середині розрахункових алгоритмів, визначити її походження буває досить складно. А якщо і вдається виявити, що це помилка в коректності вхідних даних, може бути важко показати повідомлення про помилку, бо дані могли бути введені в іншій частині програми деякий період часу тому. А якщо перевірка відбувається одразу після введення даних, то складнощів з ідентифікацією джерела проблеми не виникає.

Серед вхідних даних для розрахунків в розробленій системі є багато величин, які потребують перевірки перед потраплянням до основних розрахункових алгоритмів, наприклад глибина, частота або густина не можуть бути від’ємними величинами, тому в розробленій системі валідація даних відбувається двічі — на стороні клієнта та на стороні сервера. Перевірка на стороні клієнта відбувається при підтвердженні відправлення форми, перед відправленням на сервер, це зменшує об’єм мережевого трафіку та навантаження на сервер. Валідація на сервері потрібна тому, що недобросовісні користувачі можуть, використовуючи додаткове програмне забезпечення, надіслати дані не використовуючи форму вводу, таким чином оминувши валідацію на клієнтській частині. Також в разі розширення системи та надання доступу до API стороннім додаткам неможливо забезпечити правильно проведення валідації на стороні клієнта, тому перевірка даних на стороні серверу є важливою.

5. МЕТОДИКА РОБОТИ КОРИСТУВАЧА З СИСТЕМОЮ

5.1. Вимоги до функціонування програмного продукту

Запуск програмного продукту можна здійснювати двома способами: використовуючи засіб віртуалізації Docker або розгортати власне в операційній системі.

Для того, щоб виконати програму користуючись Docker, необхідно попередньо встановити Docker Engine. У такому випадку не потрібно встановлювати ніяких інших сторонніх засобів. Усі бібліотеки та залежності системи будуть розгортатись автоматично в Docker контейнері.

Для розгортання в операційній системі необхідно попередньо встановити наступні компоненти:

— .NET Core SDK версії не менше 3.1.100, якщо користувач бажає не тільки запускати, а й перекомпільовувати проект в своєму середовищі. ASP.NET Core Runtime 3.1.0, якщо необіхдно виконати тільки запуск;

— менеджер пакунків npm.

Для коректного відображення графічного інтерфейсу необхідно використовувати браузер з підтримкою HTML5.

5.2. Сценарій роботи користувача з системою

Після запуску системи, користувачу відкривається головна сторінка (рисунок 5.1), яка містить форму для вводу вхідних даних та навігаційне меню. Поля форми описують акустичну проблему, яку потрібно розв’язати. У плейсхолдерах елементів форми, які потребують складну структуру наведено приклад правильного формату введення даних. Щоб відправити дані з форми на сервер потрібно натиснути кнопку “Submit”, яка знаходиться в лівому нижньому куті форми.

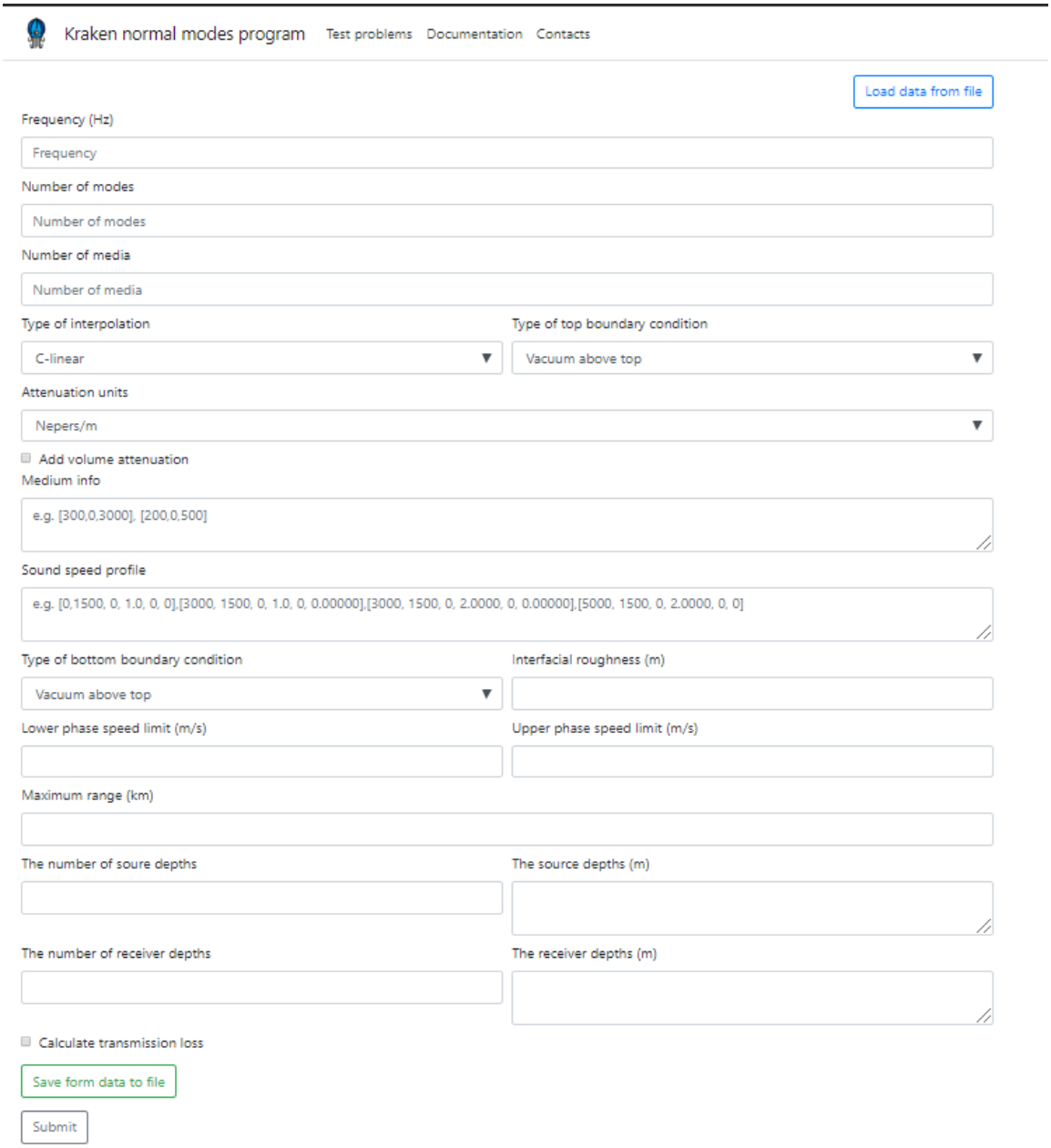


Рисунок 5.1 — Головна сторінка веб-сервісу

У разі виникнення помилок при перевірці коректності вхідних даних, користувачу відображається повідомлення з переліком помилок (рисунок 5.2).

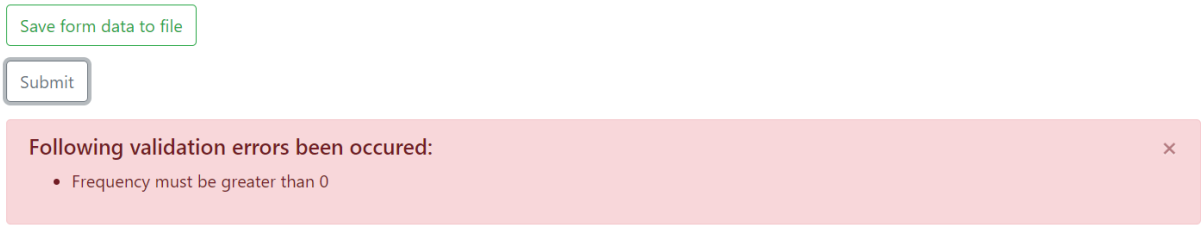


Рисунок 5.2 — Повідомлення про некоректність вхідних даних

Користувачу надається можливість не заповнювати поля вручну, а завантажити дані з файлу. Для цього потрібно натиснути кнопку “Load data from file”, яка міститься в правому верхньому куті форми, після чого відкриється діалогове вікно вибору файлу (рисунок 5.3).

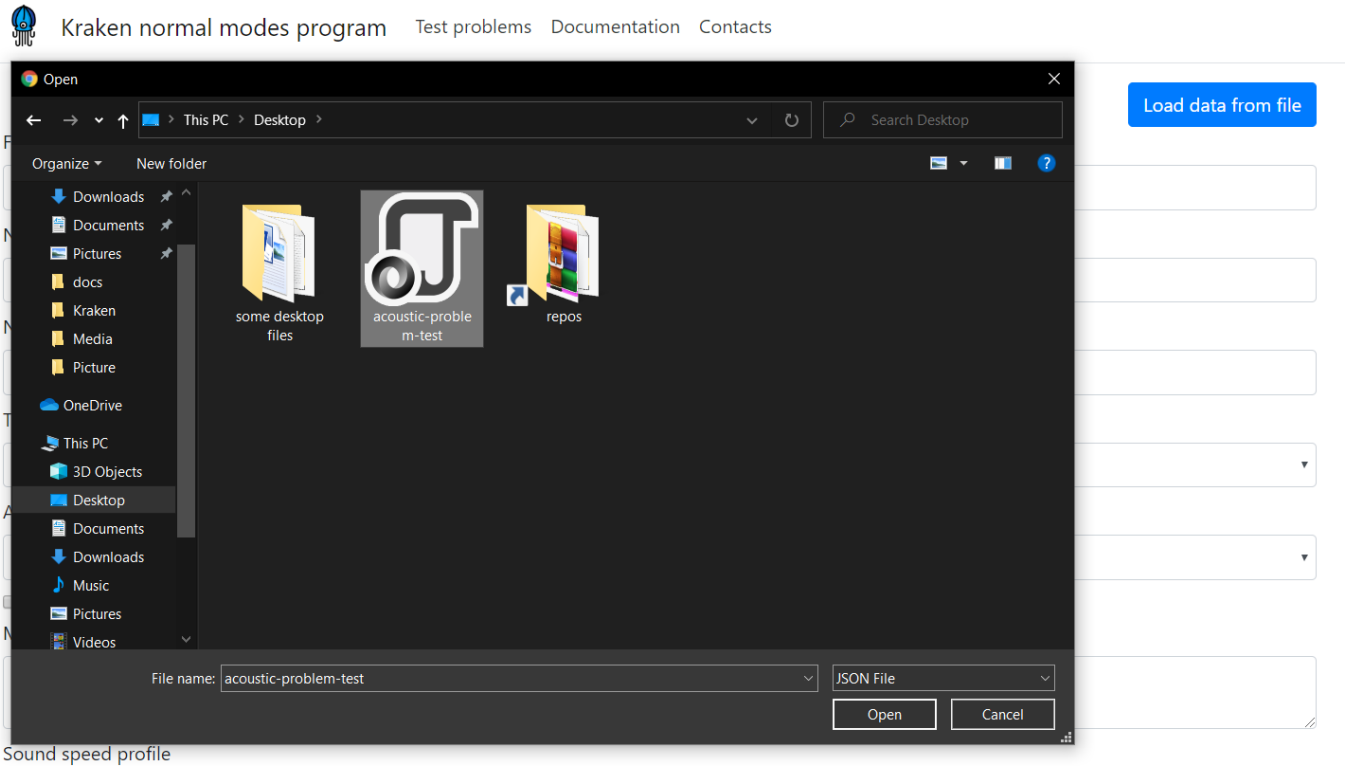


Рисунок 5.3 — Діалогове вікно вибору файлу з вхідними даними

У випадку виникнення помилок при завантаженні файлів, користувачу відображається повідомлення про помилку, наприклад на рисунку 5.4 зображено повідомлення про помилку, у випадку, коли користувач обере файл, який не містить даних у форматі JSON.

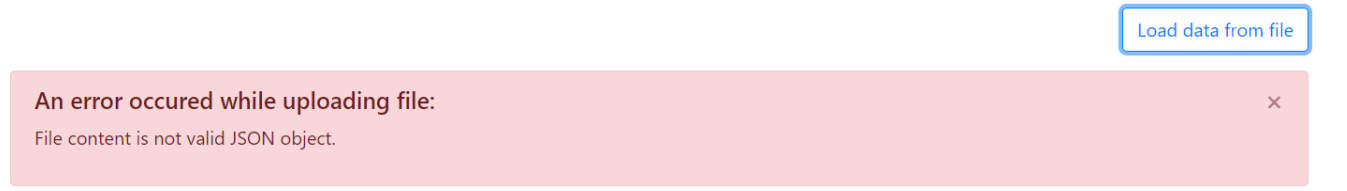


Рисунок 5.4 — Повідомлення про помилку при завантаженні файлу

В оригінальній програмі KRAKEN файли з вхідними даними можна використовувати повторно. В розробленій системі є функціонал для збереження даних з форми в локальний файл. Для цього потрібно натиснути кнопку “Save form data to file”, яка знаходиться в лівому нижньому куті форми. Після цього відкриється діалогове вікно (рисунок 5.5) для збереження файлу типу JSON з вхідними даними.

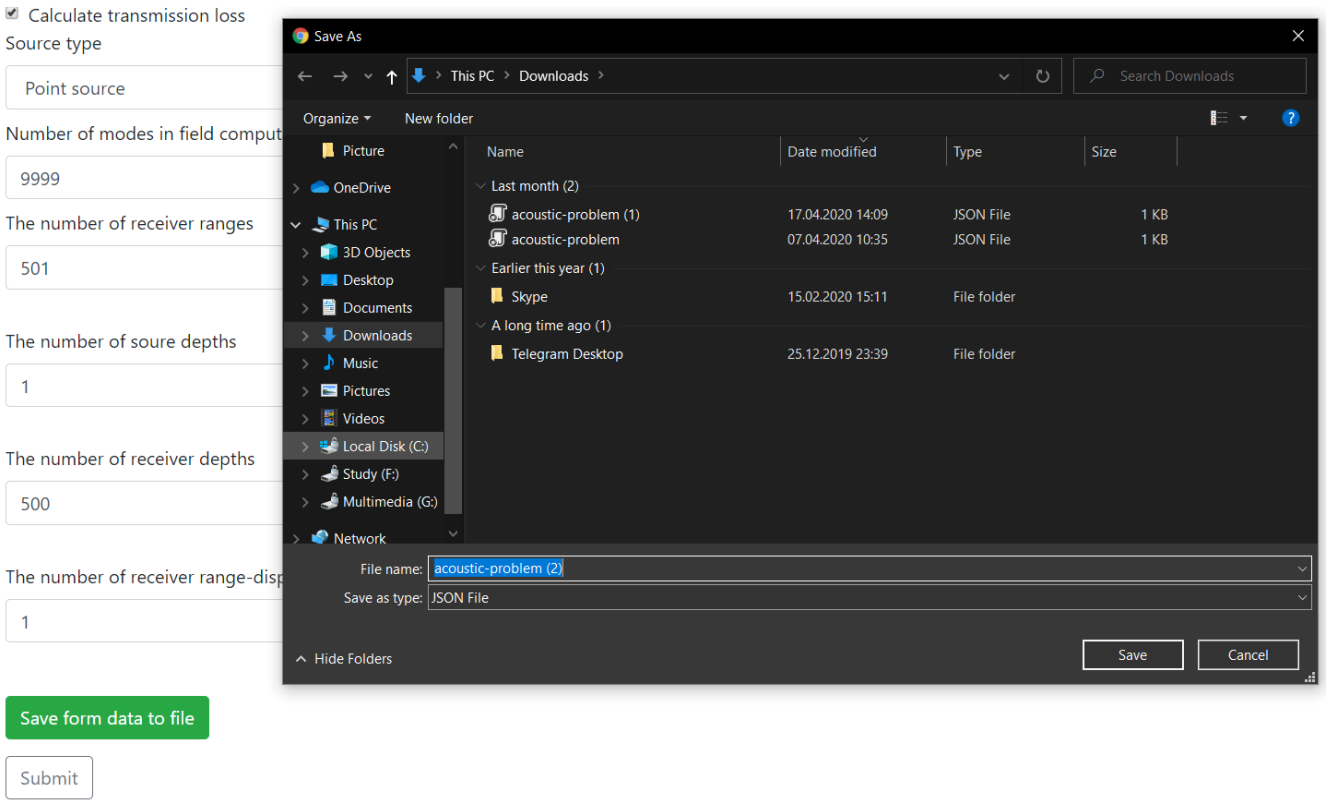


Рисунок 5.5 — Діалогове вікно завантаження файлу з вхідними данними

У разі успішного виконання програми, на веб-сторінці під формою відображається результат обрахунків у вигляді таблиць та графіків. Кожну таблицю можна зберегти у наступних форматах: JSON, XML, *.csv, *.xls. Для цього потрібно натиснути відповідні кнопки, які знаходяться у правому верхньому куту біля кожної таблиці.

Результати поділені на секції. В першій секції відображається таблиця (рисунок 5.6) з підрахованим хвильовим числом, втратою розсіювання, груповою та фазовою швидкістю для кожної з мод. Для кожної з величин побудовано графік,

наприклад на рисунку 5.7 зображено залежність величини хвильового числа від номера моди.

Collapse **Wavenumber, scatter loss, phase and group speed for each mode**

Save as JSON | Save as XML | Save as .csv | Save as .xls

#	k (1/m)	alpha (1/m)	Phase speed (m/s)	Group speed (m/s)
1	0.0635970212364862	0	1481.9527357615134	1480.1989944186892
2	0.06346151437950932	0	1485.1170907154533	1480.3473122081828
3	0.06332987703065657	0	1488.2040519685606	1480.439009306741
4	0.06320050172915292	0	1491.2504968962844	1480.440110661789
5	0.06307084943435023	0	1494.3160026058517	1480.270069680279
6	0.06294131556605655	0	1497.391320154109	1480.4349712927517
7	0.06281308264457376	0	1500.4482448503995	1480.5478513186845
8	0.06268659652274626	0	1503.4757800815576	1480.8150436610765
9	0.0625615402329161	0	1506.4811265325322	1481.1292556937715
10	0.06243827591618656	0	1509.4551895412105	1481.4664782749103
11	0.06231684217591761	0	1512.3965900203445	1481.871901405118

Рисунок 5.6 — Таблиця залежнос хвильового числа, втрати розсіювання, групової та фазової швидкості від номеру моди

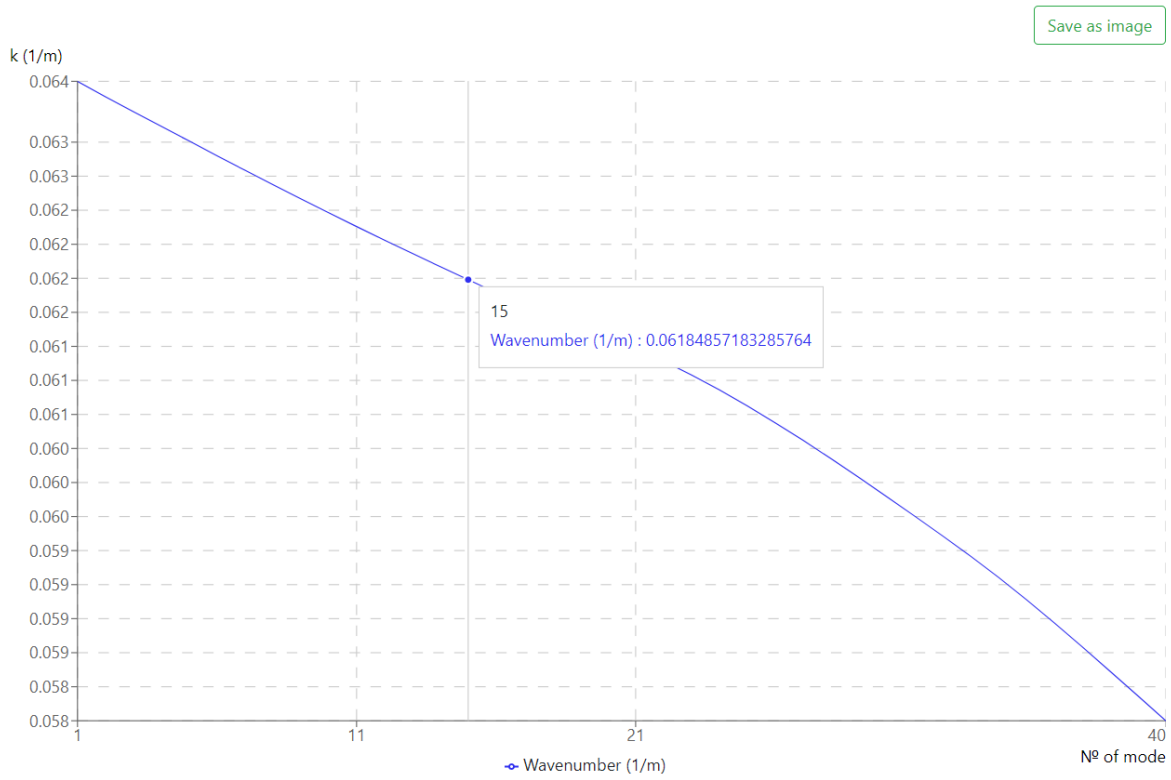
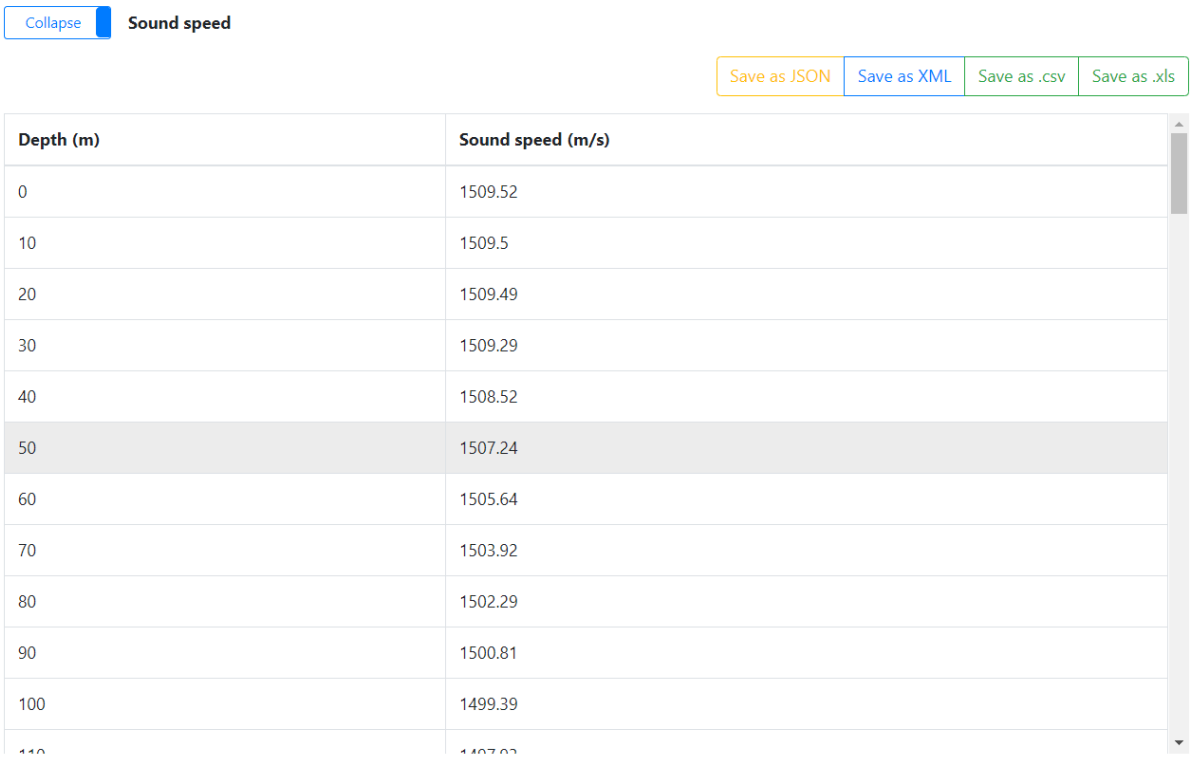


Рисунок 5.7 — Графік залежності хвильового числа від номеру моди

Наступна секція відображає у вигляді таблиці (рисунок 5.8) та графіку (рисунок 5.9) залежність звуку від глибини.



Depth (m)	Sound speed (m/s)
0	1509.52
10	1509.5
20	1509.49
30	1509.29
40	1508.52
50	1507.24
60	1505.64
70	1503.92
80	1502.29
90	1500.81
100	1499.39

Рисунок 5.8 — Таблиця залежності швидкості звуку від глибини

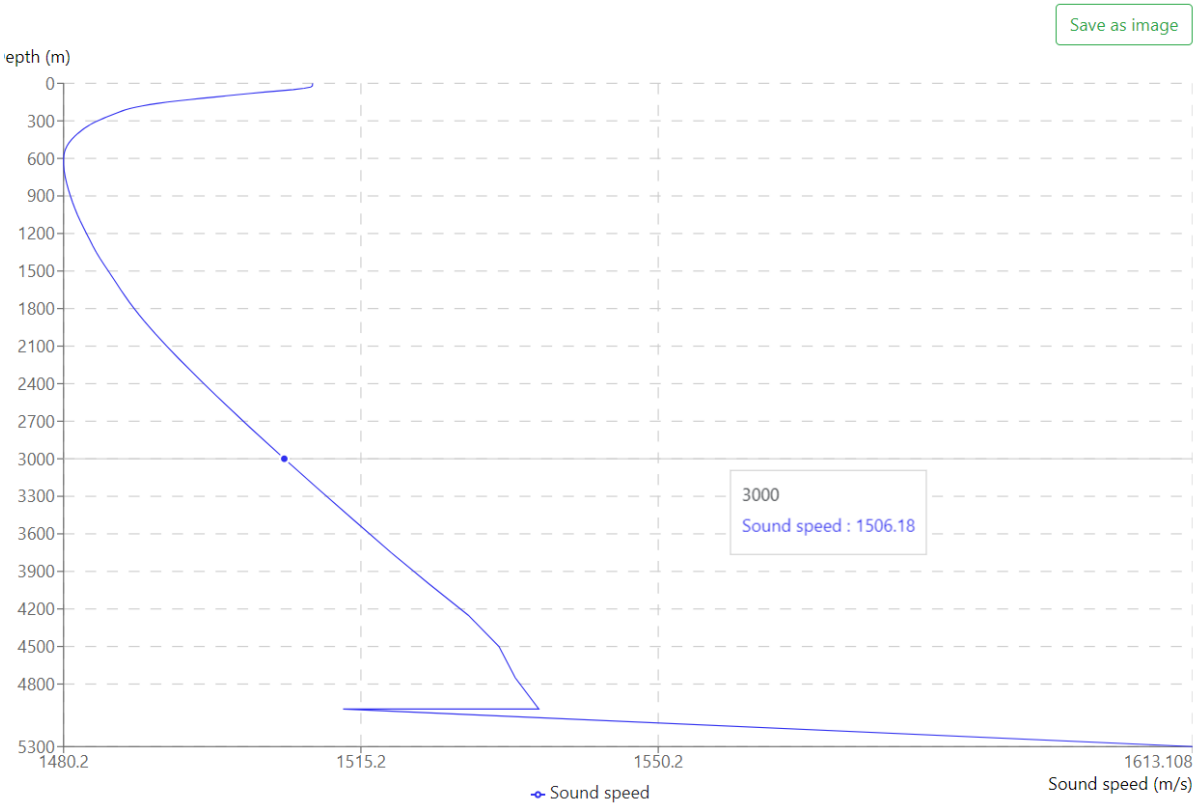


Рисунок 5.9 — Графік залежності глибини звуку від глибини

Третя секція призначеня для відображення у вигляді таблиці (рисунок 5.10) та графіку (рисунок 5.11) залежності нормальних мод від глибини.

Save as JSON | Save as XML | Save as .csv | Save as .xls

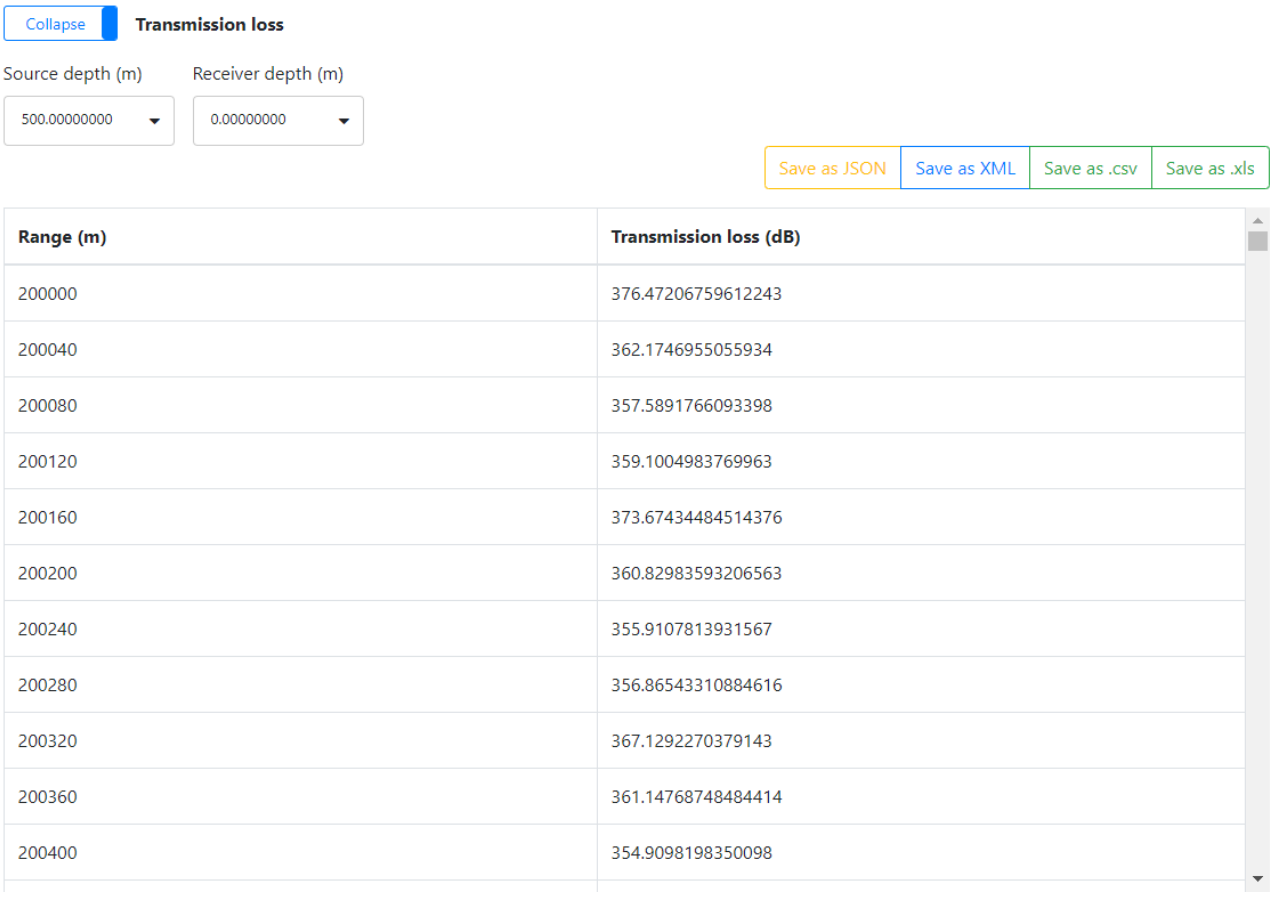
	Depth (m)				
Number of mode	**0.000**	**10.621**	**21.242**	**31.864**	**42.485**
1	1.2031990791843657e-16	0.00022351969115770674	0.0004507440453822996	0.0006853785177535405	0.0009311154046627929
2	-6.279734922204718e-18	0.0003981753909381869	0.0008021725341416277	0.0012178007169118172	0.0016507988265824275
3	8.561379173595262e-18	0.0005859052499168617	0.0011792680708461467	0.0017875121175211792	0.0024178989838723654
4	-1.793417292553683e-17	0.0007783332241283801	0.0015651288146962734	0.0023687891452421877	0.003197435901874123
5	2.919704675794523e-17	0.0009747512160036936	0.001958291713059978	0.0029593235665783023	0.003986128739581601
6	-4.906918325382394e-17	0.0011398986237496204	0.0022879675017794527	0.0034522685501204714	0.004640307874143019

Рисунок 5.10 — Таблиця залежості амплітуд нормальних мод від глибини

Рисунок 5.11 — Графік залежості амплітуд нормальних мод від глибини

При розв’язанні деяких проблем, кількість нормальних мод може бути надто великою, щоб відобразити їх всі на графіку. Тому додано можливість обирати конкретно, які моди відображати на графіку.

Остання секція містить таблицю (рисунок 5.12) та графік (рисунок 5.13) з відображенням втрати передачі на заданій глибині джерела та приймача.



Range (m)	Transmission loss (dB)
200000	376.47206759612243
200040	362.1746955055934
200080	357.5891766093398
200120	359.1004983769963
200160	373.67434484514376
200200	360.82983593206563
200240	355.9107813931567
200280	356.86543310884616
200320	367.1292270379143
200360	361.14768748484414
200400	354.9098198350098

Рисунок 5.12 — Таблиця залежності втрати передачі на заданій глибині джерела та приймача

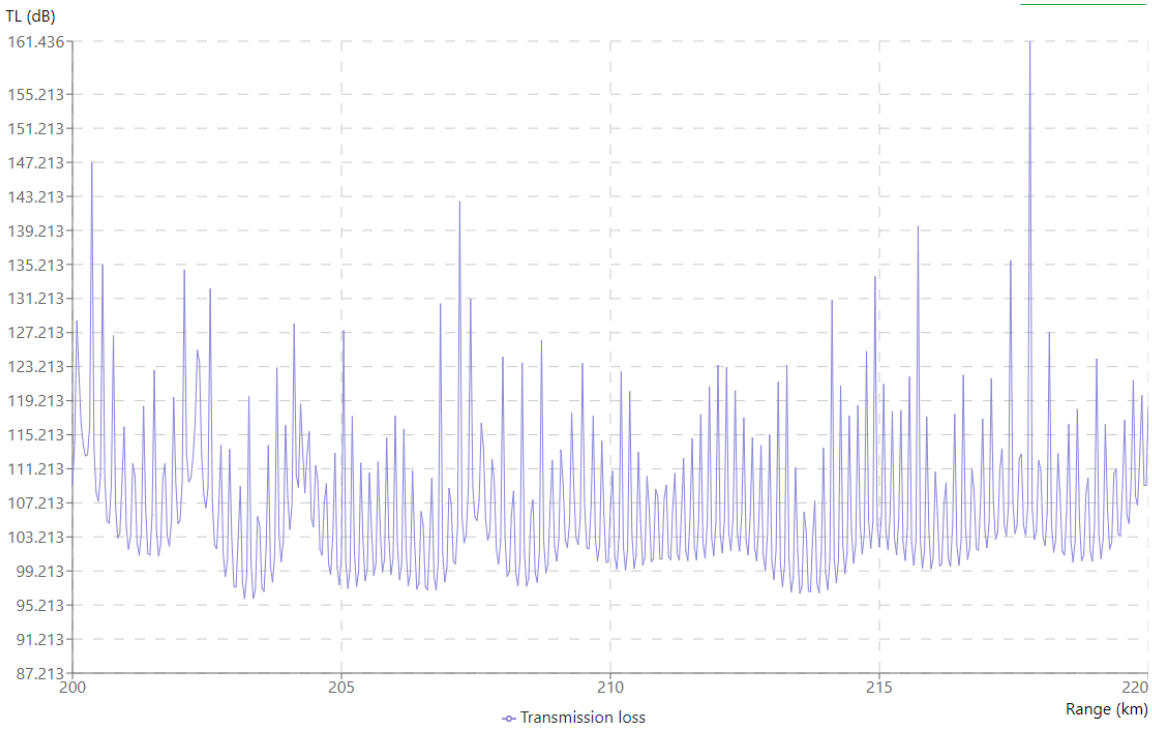


Рисунок 5.13 — Графік залежності втрати передачі на заданій глибині джерела та приймача

В системі реалізована сторінка з документацією, фрагмент якої зображено на рисунку 5.14. На ній описано поля вводу, їх формат, структуру файлу для завантаження.

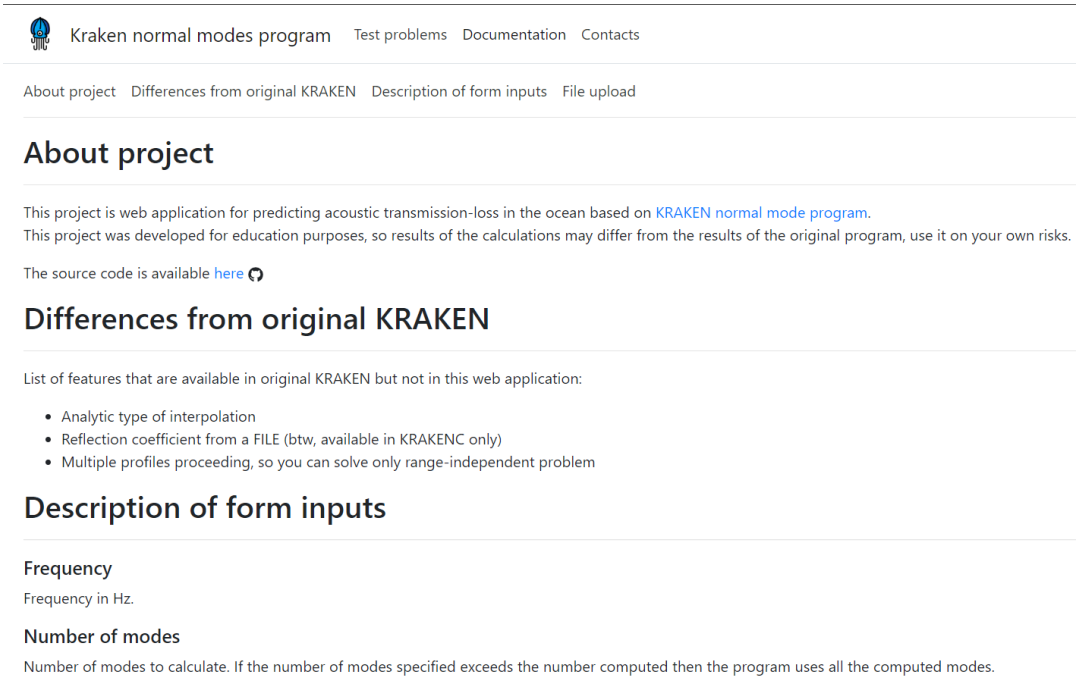


Рисунок 5.14 — Фрагмент сторінки з документацією

Також веб-сервіс має заздалегідь підготовлені тестові задачі (рисунок 5.15), які використовуються для демонстрації та тестування правильності роботи системи.

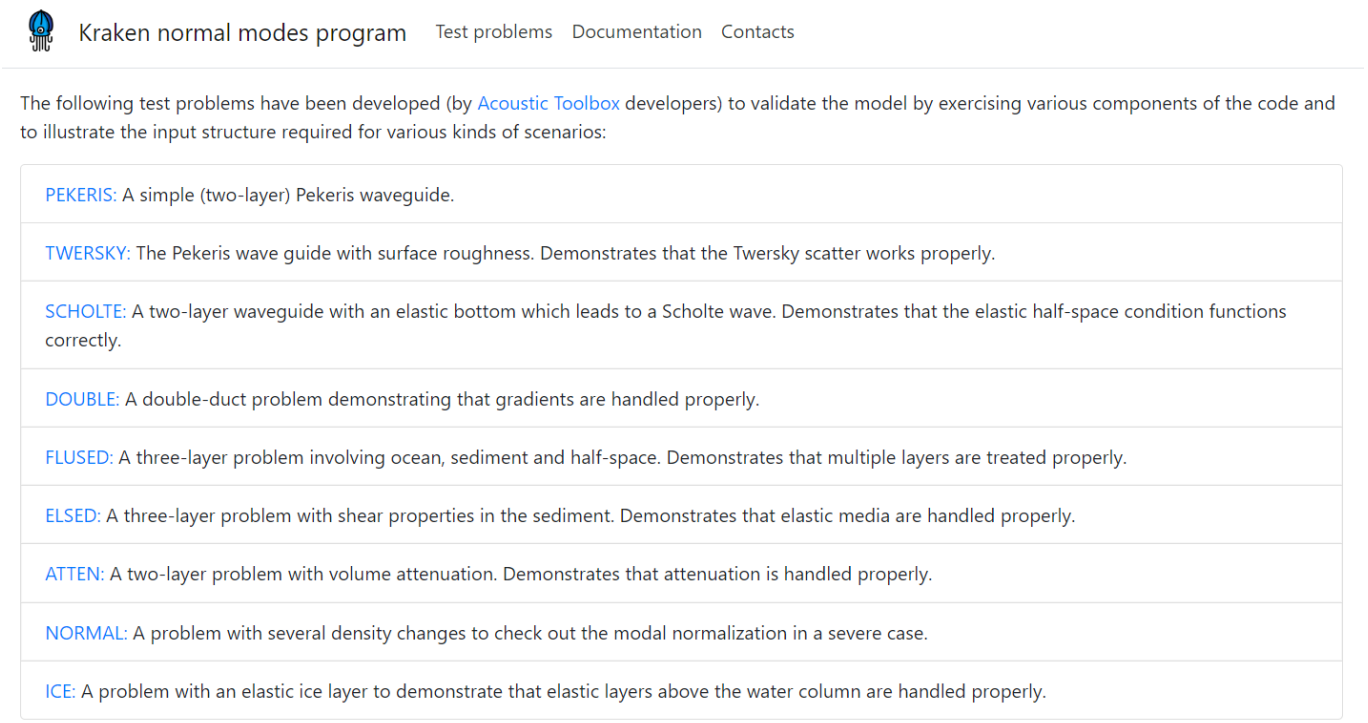


Рисунок 5.15 — Сторінка зі списком тестових задач

Дані для тестових задач взяті з відкритих кодів інших наявних систем, розроблених на основі KRAKEN. Перед списком тестових задач наведено посилання на першоджерела.

ВИСНОВКИ

В результаті бакалаврської роботи було розроблено веб-сервіс для гідроакустичних розрахунків на основі моделей KRAKEN.

Було проаналізовано програму KRAKEN та існуючі системи побудовані на її основі. При дослідженні існуючих програмних продуктів було виялено переваги та недоліки існуючих програмних продуктів. На основі проведених досліджень існучих систем, не було знайдено системи, яка працює як веб-застосунок або надає графічний інтерфейс для введення даних. На основі можливостей систем аналогів, перед початком кодування, було сформовано перелік функціональних та технічних вимог.

На підставі висунутих вимог було обрано засоби та технології розробки. Під час розробки веб-сервісу було пройдено основі етапи життєвого циклу програмного продукту, а саме: специфікація та аналіз вимог, кодування, розгортання. Розроблений веб-застосунок має клієнт-серверну архітектуру та розроблений з використанням мов програмування C# та JavaScript, фреймворку ASP.NET Core та бібліотеки React.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. M.B. Porter, The KRAKEN Normal Mode Program (No. NRL/MR/5120-92-6920), Naval Research Lab., Washington DC, 1992 207 pp..

2. C. L. Pekeris, ``Theory of propagation of explosive sound in shallow water," Geol. Soc. Amer. Mem. 27 (1948).

3. A. O. Williams, ``Normal-mode methods in propagation of underwater sound," in Underwater Acoustics, ed. R. W. B. Stephens, (Wiley-Interscience, New York, 1970).

4. AT [Електронний ресурс] — Режим доступу: https://oalib-acoustics.og/AcousitcToolbox/index_at.html

5. Normal Mode Models [Електронний ресурс] — Режим доступу: https://oalib-acoustics.org/Modes/index.html

6. Visual Studio IDE, Code Editor, Azure DevOps, & App Center - Visual Studio [Електронний ресурс] — Режим доступу: https://visualstudio.microsoft.com

7. C# 7.0. Справочник. Полное описание языка, 7-е издание / Джозеф Албахари, Бен Албахари — Диалектика, 2016 — 1088с.

8. What is .NET Core [Електронний ресурс] — Режим доступу: https://www.educba.com/what-is-dot-net-core/

9. Introduction to ASP.NET Core [Електронний ресурс] — Режим доступу: https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core

10. React – A JavaScript library for building user interfaces [Електронний ресурс] — Режим доступу: https://reactjs.org/

11. Что такое Virtual DOM? [Електронний ресурс] — Режим доступу: https://habr.com/ru/post/256965/.

12. Introducing JSX [Електронний ресурс] — Режим доступу: https://reactjs.org/docs/introducing-jsx.html

13. Using Docker: Developing and Deploying Software with Containers 1st Edition / Adrian Mouat — O'Reilly Media, 2016 — 354с.

14. Що таке Docker і як використовувати його з Python [Електронний ресурс] — https://codeguida.com/post/1837

15. What is Client-Server Architecture? [Електронний ресурс] — Режим доступу: https://www.w3schools.in/what-is-client-server-architecture/

16. Inversion of Control Containers and the Dependency Injection pattern [Електронний ресурс] — Режим доступу: https://martiflower.com/articles/injection.html

17. JSON [Електронний ресурс] — Режим доступу: https://www.json.org/json-en.html

ДОДАТОК 1

Веб-сервіс гідроакустичних розрахунків за моделями системи KRAKEN

Специфікація

УКР.НТУУ”КПІ”ІМ.ІГОРЯ_СІКОРСЬКОГО_ТЕФ_АПЕПС_ТВ6125_20Б

Аркушів 1

Київ — 2020

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ”КПІ”ІМ.ІГОРЯ_СІКОРСЬКОГО_ТЕФ_АПЕПС_ТВ6125_20Б 81-1	Пояснювальна записка.docx	Пояснювальня записка
Компоненти		
УКР.НТУУ”КПІ”ІМ.ІГОРЯ_СІКОРСЬКОГО_ТЕФ_АПЕПС_ТВ6125_20Б 12-1	KrakenController.cs, IKrakenService.cs, KrakenService.cs, IKrakenNormalModesProgram.cs KrakenNormalModesProgram.cs, IFieldProgram.cs, FieldProgram.cs	Основні компоненти
УКР.НТУУ”КПІ”ІМ.ІГОРЯ_СІКОРСЬКОГО_ТЕФ_АПЕПС_ТВ6125_20Б 13-1		Опис програмного модуля

ДОДАТОК 2

Веб-сервіс гідроакустичних розрахунків за моделями системи KRAKEN

Текст програмного модулю

УКР.НТУУ”КПІ”ІМ.ІГОРЯ_СІКОРСЬКОГО_ТЕФ_АПЕПС_ТВ6126 12-1

Аркушів 11

Київ — 2020

```
//Контролер, який приймає дані від клієнта, проводить
//валідацію, перетворює їх у модель, потрібну для сервісу
//та викликає метод для підрахунків. Повертає результат.
namespace Kraken.WebUI.Controllers
{
    public class KrakenController:Controller
    {
        private readonly IKrakenService _krakenService;

        private readonly IMapper<KrakenInputModel, AcousticProblemData> _krakenInputM
odelMapper;
        private readonly IMapper<KrakenComputingResult, KrakenResultModel> _krakenRes
ultModelMapper;

        private readonly IModelValidator<KrakenInputModel> _krakenInputModelValidator
;

        //Поля ініціалізуються завдяки механізму впровадження залежностей
        public KrakenController(IKrakenService krakenService,
                                IMapper<KrakenInputModel, AcousticProblemData> kraken
InputModelMapper,
                                IMapper<KrakenComputingResult, KrakenResultModel> kra
kenResultModelMapper,
                                IModelValidator<KrakenInputModel> krakenInputModelVal
idator)
        {
            _krakenService = krakenService;
            _krakenInputModelMapper = krakenInputModelMapper;
            _krakenResultModelMapper = krakenResultModelMapper;
            _krakenInputModelValidator = krakenInputModelValidator;
        }

        //Метод, який приймає модель від клієнта та при її коректності передає на
        //подальшу обробку. Повертає результат обчислень
        [HttpPost]
        public IActionResult ComputeNormalModes([FromBody] KrakenInputModel model)
        {
            if(model == null)
            {
                return BadRequest("Input data is not valid");
            }

            //Валідація даних та повернення помилки, якщо вхідні дані не валідні
            var errors = _krakenInputModelValidator.Validate(model);
            if (errors.Any())
            {
                return BadRequest(new { validationErrors= errors });
            }

            //Перетворення моделі представлення в модель, яку приймає на вхід сервіс
            var acousticProblem = _krakenInputModelMapper.Map(model);

            //Виклик сервісу та повернення результату, або помилки з описом проблем
            try
            {
                var computingResult = _krakenService.ComputeModes(acousticProblem);

                var viewModel = _krakenResultModelMapper.Map(computingResult);

                return Json(viewModel);
            }
            catch(KrakenComputingException ex)
            {
                return StatusCode(500, new { expectedError=true,error=ex.Message });
```

```
            }
        }
    }
}

namespace Kraken.Application.Services.Interfaces
{
    //Інтерфейс IKrakenService декларує метод, який повинен релізовувати клас, який
    //буде проводити обчислення та повертати результат
    public interface IKrakenService
    {
        KrakenComputingResult ComputeModes(AcousticProblemData acousticProblemData);
    }
}

namespace Kraken.Application.Services.Implementation
{
    //Класс KrakenService реалізує інтерфейс IKrakenService. Викликає методи
головного модуля
    //для проведення обчислень
    public class KrakenService : IKrakenService
    {
        private readonly IKrakenNormalModesProgram _krakenNormalModeProgram;
        private readonly IFieldProgram _fieldModel;

        private     readonly     IMapper<AcousticProblemData,     KrakenInputProfile>
_krakenInputProfileMapper;
        private    readonly    IMapper<FieldComputingRequiredData,    FieldInputData>
_fieldInputDataMapper;
        private        readonly        IMapper<KrakenResultAndAcousticFieldSnapshots,
KrakenComputingResult> _krakenComputingResultMapper;

        //Поля ініціалізуються завдяки механізму впровадження залежностей
        public KrakenService(IKrakenNormalModesProgram krakenNormalModeProgram,
            IFieldProgram fieldModel,
            IMapper<AcousticProblemData,                              KrakenInputProfile>
krakenInputProfileMapper,
            IMapper<FieldComputingRequiredData, FieldInputData> fieldInputDataMapper,
            IMapper<KrakenResultAndAcousticFieldSnapshots,     KrakenComputingResult>
krakenComputingResultMapper)
        {
            _krakenNormalModeProgram = krakenNormalModeProgram;
            _fieldModel = fieldModel;
            _krakenInputProfileMapper = krakenInputProfileMapper;
            _fieldInputDataMapper = fieldInputDataMapper;
            _krakenComputingResultMapper = krakenComputingResultMapper;
        }

        //Метод, що проводить розрахунки

        public         KrakenComputingResult         ComputeModes(AcousticProblemData
acousticProblemData)
        {
            CalculatedModesInfo modesInfo;
            KrakenResultAndAcousticFieldSnapshots
krakenResultAndAcousticFieldSnapshots = new KrakenResultAndAcousticFieldSnapshots();

            //Розрахунки за моделями типу KRAKEN. Повернення результату обчислень
            //KrakenResult та даних, що необхідні для моделей типу FIELD та описані
класом
            //CalculatedModesInfo
            try
            {
                //Перетворення даних в модель потрібну для основного модуля
```

```
                var profile = _krakenInputProfileMapper.Map(acousticProblemData);
                //Виклик функції за моделями KRAKEN
                (krakenResultAndAcousticFieldSnapshots.KrakenResult, modesInfo) =
                    _krakenNormalModeProgram.CalculateNormalModes(profile);
            }
            catch(KrakenException ex)
            {
                throw new KrakenComputingException(ex.Message);
            }

            //Проведення обчислень за моделями типу FIELD, якщо поле
            //CalculateTransmissionLoss має значення true - це означає, що потрібно
            //провести підрахунок втрати передачі
            if (acousticProblemData.CalculateTransmissionLoss)
            {
                var fieldComputingRequiredData = new FieldComputingRequiredData
                {
                    AcousticProblemData = acousticProblemData,
                    ModesInfo = modesInfo
                };

                //Перетворення даних в модель потрібну для основного модуля
                var                              fieldInput                                =
_fieldInputDataMapper.Map(fieldComputingRequiredData);

                //Виклик функцій за моделями FIELD
                krakenResultAndAcousticFieldSnapshots.AcousticFieldSnapshots        =
_fieldModel.CalculateFieldPressure(fieldInput);

                //Обчислення втрати передачі на основі знімків акустичного поля
                krakenResultAndAcousticFieldSnapshots.TransmissionLoss.AddRange(

CalculateTransmissionLossUsingAcousticSnapshots(krakenResultAndAcousticFieldSnapshots
.
                    AcousticFieldSnapshots.Snapshots)
                );
            }

            var                              result                                    =
_krakenComputingResultMapper.Map(krakenResultAndAcousticFieldSnapshots);

            return result;
        }

        //Метод для обчислення втрати передачі на основі знімків акустичного поля
        private                                          IEnumerable<List<List<double>>>
CalculateTransmissionLossUsingAcousticSnapshots(List<List<List<Complex>>> snapshots)
        {
            return snapshots.GetRange(1, snapshots.Count - 1)
                            .Select(x => x.GetRange(1, x.Count - 1)
                            .Select(y => y.GetRange(1, y.Count - 1)
                            .Select(z => z.Real == 0 ? 1E-6 : z.Real)
                            .Select(z               =>              -20              *
Math.Log10(Math.Abs(z))).ToList()).ToList());
        }
    }
}

namespace Kraken.Calculation.Interfaces
{
    //Інтерфейс  IKrakenNormalModesProgram,описує  поведінку  класа,  який  повинен
реалізовувати
    //обчислення за моделями типу KRAKEN
    public interface IKrakenNormalModesProgram
```

```
    {
        (KrakenResult, CalculatedModesInfo) CalculateNormalModes(KrakenInputProfile
profile);
    }
}

namespace Kraken.Calculation
{
    //Клас, що проводить розрахунки за моделями типу KRAKEN. В своїх методах
    //використовує функціонал другорядних класів
    public class KrakenNormalModesProgram : IKrakenNormalModesProgram
    {
        //Головний метод
        public                    (KrakenResult,                    CalculatedModesInfo)
CalculateNormalModes(KrakenInputProfile profile)
        {
            //Ініціалізація, зчитування вхідної інформації та, за потреби, її
перетворення
            var krakenModule = new KrakenModule();
            krakenModule.Init();

            krakenModule.NV = new List<int> { 0, 1, 2, 4, 8, 16 };

            krakenModule.Frequency = profile.Frequency;
            krakenModule.NMedia = profile.NMedia;
            krakenModule.BCTop = profile.Options.Substring(0, 3);
            krakenModule.BCBottom = profile.BCBottom;

            var rangedDataManager = new RangedDataManager();
            var meshInitializer = new MeshInitializer(krakenModule);

            krakenModule.Depth[1] = 0;
            for (var media = 1; media <= profile.NMedia; media++)
            {
                krakenModule.NG[media] = (int)profile.MediumInfo[media][1];
                krakenModule.Sigma[media] = profile.MediumInfo[media][2];
                krakenModule.Depth[media + 1] = profile.MediumInfo[media][3];
            }

            krakenModule.Sigma[profile.NMedia + 1] = profile.BottomSigma;
            //обробка SSP
            meshInitializer.ProccedMesh(krakenModule,                    profile.SSP,
profile.TopAcousticHSProperties,                    profile.TwerskyScatterParameters,
profile.BottomAcousticHSProperties);

            krakenModule.CLow = profile.CLow;
            krakenModule.CHigh = profile.CHigh;

            krakenModule.RMax = profile.RMax;

            var zMin = krakenModule.Depth[1];
            var zMax = krakenModule.Depth[krakenModule.NMedia + 1];

            //обробка глибин джерела та приймача
            rangedDataManager.ProceedSourceAndReceiverDepths(zMin, zMax, profile.Nsd,
profile.Nrd, profile.SourceDepths, profile.ReceiverDepths);

            krakenModule.Omega2 = Math.Pow((2.0 * Math.PI * krakenModule.Frequency),
2);

            double error = 0;
            var isSuccess = false;
            var modesInfo = new CalculatedModesInfo();
```

```
            var zm = new List<double>();
            var modes = new List<List<double>>();

            //Багаторазовий підрахунок для збільшення точності
            for (krakenModule.ISet = 1; krakenModule.ISet <= krakenModule.NSets;
krakenModule.ISet++)
            {
                krakenModule.N = krakenModule.NG.Select(x => x *
krakenModule.NV[krakenModule.ISet]).ToList();
                for (var j = 1; j <= krakenModule.NMedia; j++)
                {
                    krakenModule.H[j] = (krakenModule.Depth[j + 1] -
krakenModule.Depth[j]) / krakenModule.N[j];
                }
                krakenModule.HV[krakenModule.ISet] = krakenModule.H[1];
                Solve(modesInfo, krakenModule, rangedDataManager, meshInitializer,
ref error, profile.NModes, ref zm, ref modes);

                //зупинка про достатній точності результату
                if (error * 1000.0 * krakenModule.RMax < 1.0)
                {
                    isSuccess = true;
                    break;
                }
            }

            var result = new KrakenResult();

            if (!isSuccess)
            {
                result.Warnings.Add("Too many meshes needed: check convergence");
            }

            //Приведення результату до потрібної форми
            var omega = Math.Sqrt(krakenModule.Omega2);
            var minVal = krakenModule.Extrap[1].GetRange(1, krakenModule.M).Where(x
=> x > krakenModule.Omega2 / Math.Pow(krakenModule.CHigh, 2)).Min();
            var minLoc = krakenModule.Extrap[1].FindIndex(x => x == minVal);
            krakenModule.M = minLoc;

            for (var i = 1; i <= krakenModule.M; i++)
            {
                krakenModule.K[i] = Complex.Sqrt(krakenModule.Extrap[1][i] +
krakenModule.K[i]);
            }

            var MMM = Math.Min(krakenModule.M, profile.NModes);

            modesInfo.ModesCount = MMM;
            modesInfo.K.AddRange(krakenModule.K);

            var cp = Enumerable.Repeat(0d, MMM + 1).ToList();
            var cg = Enumerable.Repeat(0d, MMM + 1).ToList();
            var k = Enumerable.Repeat(new Complex(), MMM + 1).ToList();

            for (krakenModule.Mode = 1; krakenModule.Mode <= MMM;
krakenModule.Mode++)
            {
                cp[krakenModule.Mode] = (omega /
krakenModule.K[krakenModule.Mode]).Real;
                cg[krakenModule.Mode] = krakenModule.VG[krakenModule.Mode];
                k[krakenModule.Mode] = krakenModule.K[krakenModule.Mode];
            }
```

```
            result.GroupSpeed.AddRange(cg);
            result.PhaseSpeed.AddRange(cp);
            result.K.AddRange(k);
            result.ModesCount = MMM;
            result.Modes.AddRange(modes);
            result.ZM.AddRange(zm);

            result.Warnings.AddRange(krakenModule.Warnings);

            return (result, modesInfo);
        }
//Підрахунок нормальних мод
        private void Vector(CalculatedModesInfo modesInfo, KrakenModule krakenModule,
RangedDataManager rangedDataManager, int nm, ref List<double> zm, ref
List<List<double>> modes)
        {
            var BCTop = krakenModule.BCTop[1].ToString();
            var BCBottom = krakenModule.BCBottom[0].ToString();

            var NTot = krakenModule.N.GetRange(krakenModule.FirstAcoustic,
krakenModule.LastAcoustic - krakenModule.FirstAcoustic + 1).Sum();
            var NTot1 = NTot + 1;

            var z = Enumerable.Repeat(0d, NTot1 + 1).ToList();
            var e = Enumerable.Repeat(0d, NTot1 + 1 + 1).ToList();
            var d = Enumerable.Repeat(0d, NTot1 + 1).ToList();
            var Phi = Enumerable.Repeat(0d, NTot1 + 1).ToList();
            var j = 1;
            z[1] = krakenModule.Depth[krakenModule.FirstAcoustic];

            var hRho = 0.0;
            for (var medium = krakenModule.FirstAcoustic; medium <=
krakenModule.LastAcoustic; medium++)
            {
                hRho = krakenModule.H[medium] *
krakenModule.Rho[krakenModule.Loc[medium] + 1];

                var temp = 1;
                for (var i = j + 1; i <= j + krakenModule.N[medium]; i++)
                {
                    e[i] = 1.0 / hRho;
                    z[i] = z[j] + krakenModule.H[medium] * temp;
                    temp++;
                }

                j += krakenModule.N[medium];
            }

            e[NTot1 + 1] = 1.0 / hRho;
            var vectorsManager = new VectorsManager();
            var (zTab, NzTab) =
vectorsManager.MergeVectors(rangedDataManager.SourceDepths, rangedDataManager.Nsd,
rangedDataManager.ReceiverDepths, rangedDataManager.Nrd);

            var PhiTab = Enumerable.Repeat(new Complex(), NzTab + 1).ToList();

            var weightsCalculator = new WeightsCalculator();
            var (weights,indicesTab) =
weightsCalculator.CalculateWeightsAndIndices(z, NTot1, zTab, NzTab);

            var modesave = new List<List<double>>(NzTab + 1);
            for (var i = 0; i <= NzTab + 1; i++)
            {
                modesave.Add(Enumerable.Repeat(0d, krakenModule.M + 1).ToList());
```

```
            }

            //запис даних, необхідних для моделей типу Field
            modesInfo.NMedia = krakenModule.LastAcoustic - krakenModule.FirstAcoustic
+ 1;
            modesInfo.NTot = NzTab;
            modesInfo.NMat = NzTab;

            modesInfo.N.AddRange(krakenModule.N);

            modesInfo.Material.AddRange(Enumerable.Repeat("",
krakenModule.Material.Count));

            modesInfo.Depth.AddRange(Enumerable.Repeat(0d,
krakenModule.Depth.Count));

            modesInfo.Rho.AddRange(Enumerable.Repeat(0d, krakenModule.Rho.Count));

            for    (var    medium    =    krakenModule.FirstAcoustic;    medium    <=
krakenModule.LastAcoustic; medium++)
            {
                modesInfo.Material[medium] = krakenModule.Material[medium];
                modesInfo.Depth[medium] = krakenModule.Depth[medium];
                modesInfo.Rho[medium]  =  krakenModule.Rho[krakenModule.Loc[medium]  +
1];
            }

            modesInfo.Frequency = krakenModule.Frequency;
            modesInfo.Z.AddRange(zTab);

            modesInfo.BCTop = krakenModule.BCTop[1].ToString();
            modesInfo.CPTop = krakenModule.CPTop;
            modesInfo.CSTop = krakenModule.CSTop;
            modesInfo.RhoTop = krakenModule.RhoTop;
            modesInfo.DepthTop = krakenModule.Depth[1];

            modesInfo.BCBottom = krakenModule.BCBottom[0].ToString();
            modesInfo.CPBottom = krakenModule.CPBottom;
            modesInfo.CSBottom = krakenModule.CSBottom;
            modesInfo.RhoBottom = krakenModule.RhoBottom;
            modesInfo.DepthBottom = krakenModule.Depth[krakenModule.NMedia + 1];

            var f = 0.0;
            var g = 0.0;
            var iPower = 0;
            modesInfo.Phi.Add(new List<Complex>());
            for  (krakenModule.Mode  =  1;  krakenModule.Mode  <=  krakenModule.M;
krakenModule.Mode++)
            {
                var x = krakenModule.EVMat[1][krakenModule.Mode];
                var bcimpSolver = new BCImpedanceSolver();
                bcimpSolver.ComputeBoundaryConditionImpedance(krakenModule, x, BCTop,
"TOP", krakenModule.CPTop, krakenModule.CSTop, krakenModule.RhoTop, ref f, ref g, ref
iPower);

                int l;
                double xH2;
                if (g == 0.0)
                {
                    d[1] = 1.0;
                    e[2] = 2.220446049250313 / Math.Pow(10, 16);
                }
                else
                {
```

```
                    l = krakenModule.Loc[krakenModule.FirstAcoustic] + 1;
                    xH2    =    x    *    krakenModule.H[krakenModule.FirstAcoustic]    *
krakenModule.H[krakenModule.FirstAcoustic];
                    hRho      =      krakenModule.H[krakenModule.FirstAcoustic]      *
krakenModule.Rho[l];
                    d[1] = (krakenModule.B1[l] - xH2) / hRho / 2.0 + f / g;
                }

                var iTP = NTot;
                j = 1;
                l = krakenModule.Loc[krakenModule.FirstAcoustic] + 1;

                for    (var    medium    =    krakenModule.FirstAcoustic;    medium    <=
krakenModule.LastAcoustic; medium++)
                {
                    xH2 = x * Math.Pow(krakenModule.H[medium], 2);
                    hRho            =            krakenModule.H[medium]            *
krakenModule.Rho[krakenModule.Loc[medium] + 1];

                    if (medium >= krakenModule.FirstAcoustic + 1)
                    {
                        l += 1;
                        d[j] = (d[j] + (krakenModule.B1[l] - xH2) / hRho) / 2.0;
                    }

                    for (var ii = 1; ii <= krakenModule.N[medium]; ii++)
                    {
                        j += 1;
                        l += 1;
                        d[j] = (krakenModule.B1[l] - xH2) / hRho;

                        if (krakenModule.B1[l] - xH2 + 2.0 > 0.0)
                        {
                            iTP = Math.Min(j, iTP);
                        }
                    }
                }

                bcimpSolver.ComputeBoundaryConditionImpedance(krakenModule,        x,
BCBottom,         "BOT",         krakenModule.CPBottom,         krakenModule.CSBottom,
krakenModule.RhoBottom, ref f, ref g, ref iPower);
                if (g == 0.0)
                {
                    d[NTot1] = 1.0;
                    e[NTot1] = 2.220446049250313 / Math.Pow(10, 16);
                }
                else
                {
                    d[NTot1] = d[NTot1] / 2.0 - f / g;
                }

                var errorFlag = 0;
                var sinvitMod = new EigenvectorFinder();
                //Обчислення власного вектору за допомогою методу зворотньої ітерації
                sinvitMod.FindEigenvectorUsingInverseIteration(NTot1,  d,  e,Phi,  ref
errorFlag);

                if (errorFlag != 0)
                {
                    krakenModule.Warnings.Add($"Inverse iteration failed to converge.
Mode = {krakenModule.Mode}");
                    Phi = Phi.Select(p => 0d).ToList();
                }
                else
```

```
                {
                    //нормалізація власного вектору
                    NormalizeEigenvector(krakenModule, Phi, iTP, NTot1, x);
                }

                for (var i = 1; i <= NzTab; i++)
                {
                    PhiTab[i] = Phi[indicesTab[i]] + weights[i] * (Phi[indicesTab[i]
+ 1] - Phi[indicesTab[i]]);
                }

                modesInfo.Phi.Add(new List<Complex>(PhiTab));

                for (var i = 1; i <= NzTab; i++)
                {
                    modesave[i][krakenModule.Mode] = PhiTab[i].Real;
                }
            }

            var MMM = Math.Min(krakenModule.M, nm);
            modes = new List<List<double>>(NzTab + 1);
            for (var i = 0; i <= NzTab; i++)
            {
                modes.Add(Enumerable.Repeat(0d, MMM + 1).ToList());
            }
            zm = Enumerable.Repeat(0d, NzTab + 1).ToList();

            for (var MZ = 1; MZ <= NzTab; MZ++)
            {
                for    (krakenModule.Mode    =    1;    krakenModule.Mode    <=    MMM;
krakenModule.Mode++)
                {
                    modes[MZ][krakenModule.Mode] = modesave[MZ][krakenModule.Mode];
                }
                zm[MZ] = zTab[MZ];
            }
        }

namespace Kraken.Calculation.Field.Interfaces
{
    //Інтерфейс IFieldProgram, описує поведінку класа, який повинен реалізовувати
    //обчислення за моделями типу Field
    public interface IFieldProgram
    {
        AcousticFieldSnapshots CalculateFieldPressure(FieldInputData fieldData);
    }
}


{
    //Клас, що проводить розрахунки за моделями типу Field. В своїх методах
    //використовує функціонал другорядних класів
    public class FieldProgram : IFieldProgram
    {
        //Головний метод для обчислення тиску акустичного поля
        public      AcousticFieldSnapshots      CalculateFieldPressure(FieldInputData
fieldData)
        {
            var            maxM            =            Math.Min(fieldData.ModesLimit,
fieldData.ModesInfo.ModesCount);

            string comp = "";
            if (fieldData.Options.Length > 2)
            {
```

```
                comp = fieldData.Options[2].ToString();
            }

            var rangedDataManager = new RangedDataManager();

            //обробка діапазону приймача
            rangedDataManager.ProceedReceiverRanges(fieldData.Nr,
fieldData.ReceiverRanges);

            var zMin = -3.40282347E+38;
            var zMax = 3.40282347E+38;

            //обробка глибин джерела та глибин приймача
            rangedDataManager.ProceedSourceAndReceiverDepths(zMin,              zMax,
fieldData.Nsd, fieldData.Nrd, fieldData.SourceDepths, fieldData.ReceiverDepths);

            var result = new AcousticFieldSnapshots();

            result.Ranges.AddRange(rangedDataManager.ReceiverRanges);
            result.SourceDepths.AddRange(rangedDataManager.SourceDepths);
            result.ReceiverDepths.AddRange(rangedDataManager.ReceiverDepths);

            var C = Enumerable.Repeat(new Complex(), maxM + 1).ToList();

            var   receiverDisplacements   =   Enumerable.Repeat(0d,   fieldData.Nrr   +
1).ToList();

            var Nrr = fieldData.Nrr;

            if (fieldData.Nrr != rangedDataManager.Nrd)
            {
                Nrr = rangedDataManager.Nrd;
                receiverDisplacements = Enumerable.Repeat(0d, Nrr + 1).ToList();
            }

            for (var i = 0; i < fieldData.ReceiverDisplacements.Count; i++)
            {
                receiverDisplacements[i] = fieldData.ReceiverDisplacements[i];
            }

            if (Nrr > 1)
            {
                receiverDisplacements[2] = -999.9;
            }
            if (Nrr > 2)
            {
                receiverDisplacements[3] = -999.9;
            }

            var subTabMod = new SubTabulator();

            //табуляція зміщень діапазону отримувача
            subTabMod.SubTabulate(receiverDisplacements, Nrr);

            var readModesMod = new ModesPreparationManager();

            //читання та обробка мод, підрахованих KRAKEN
            var   phiS   =   readModesMod.GetPreparedModes(fieldData.ModesInfo,   maxM,
rangedDataManager.SourceDepths, rangedDataManager.Nsd, "N", result.Warnings);
            var   phiR   =   readModesMod.GetPreparedModes(fieldData.ModesInfo,   maxM,
rangedDataManager.ReceiverDepths, rangedDataManager.Nrd, comp, result.Warnings);
            var pressureFieldCalculator = new PressueFieldCalculator();

            result.Snapshots.Add(new List<List<Complex>>());
```

```
            //цикл по глибинах джерела
            for (var IS = 1; IS <= rangedDataManager.Nsd; IS++)
            {
                for (var i = 1; i <= fieldData.ModesInfo.ModesCount; i++)
                {
                    C[i] = phiS[i][IS];
                }

                //підрахунок тиску акустичного поля, для поточної глибини джерела
                var P = pressureFieldCalculator.Evaluate(C, phiR,
rangedDataManager.Nrd, rangedDataManager.ReceiverRanges,
                                                rangedDataManager.Nr,
receiverDisplacements, fieldData.ModesInfo.K, fieldData.ModesInfo.ModesCount,
fieldData.Options);
                result.Snapshots.Add(P);
            }

            return result;
        }
    }
}
```

ДОДАТОК З

Веб-сервіс гідроакустичних розрахунків за моделями системи KRAKEN

Опис програмного модулю

УКР.НТУУ”КПІ”ІМ.ІГОРЯ_СІКОРСЬКОГО_ТЕФ_АПЕПС_ТВ6125 13-1

Аркушів 8

Київ — 2020

АНОТАЦІЯ

Додаток являє собою веб-сервіс, призначений для проведення гідроакустичних розрахунків за моделями системи KRAKEN.

Веб-сервіс має графічний користувацький інтерфейс з формою для введення даних, сторінку з документацією, сторінку з прикладами задач. Результати обчислень відображаються у вигляді таблиць та графіків, які можна завантажувати.

Обчислювальні алгоритми були реалізовані мовою програмування C#, серверна частина додатку написана на фреймворку ASP.NET Core, а клієнтська частина з використанням мови програмування JavaScript та бібліотеки React.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

Розроблена система має назву веб-сервіс для гідроакустичних розрахунків на основі моделей KRAKEN.

Запуск програмного продукту можна здійснювати двома способами: використовуючи засіб віртуалізації Docker або розгортати в операційній системі.

Для того, щоб виконати програму користуючись Docker, необхідно попередньо встановити Docker Engine. Для розгортання в операційній системі необхідно попередньо встановити наступні компоненти:

— ASP.NET Core Runtime 3.1.0;

— менеджер пакунків npm.

Програма написана мовами програмування C# та JavaScript.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Головна задача програми — надавати можливість проведення гідроакустичних розрахунків, аналогічних розрахункам, що проводяться програмою KRAKEN з високою точністю.

У результаті виконання програми можна розрахувати наступні величини:

— нормальні моди;

— хвильове число, втрату розсіювання, фазову швидкість, групову швидкість для кожної зі знайдених мод;

— втрата передачі сигналу.

Розроблена програма, використовуючи графічний інтерфейс, взаємодіє з користувачем, інформуючи про помилки у вхідних даних, помилки та неточності при обчисленнях.

Програма надає можливість аналізувати отриманий результат, використовуючи графіки й таблиці.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

В розробленій програмній системі, клієнт, який надає графічний інтерфейс, та сервер, який проводить розрахункі, реалізовані незалежно один від одного, що надає додаткову гнучкість при розробці та експлуатації.

Складові серверної частини це один проект типу ASP.NET Core Web Api та два проекти типу бібліотеки класів. Взаємодія класів між проектами реалізована з використанням шаблону впровадження залежностей. Головний модуль програми, який відповідальний за проведення розрахунків, реалізований без зовнішніх залежностей.

Складові клієнтської частини поділяються на компоненти — складові сторінок графічного інтерфейсу, сервіси — класи, які виконують функції відправлення та отримання даних з серверу, утиліти — набір класів та методів, в які винесені допоміжний функціонал.

ТЕХНІЧНІ ЗАСОБИ, ЩО ВИКОРИСТОВУЮТЬСЯ

Програмний продукт було розроблено в середовищі розробки Microsoft Visual Studio, на комп’ютері, що використовував операційну систему Windows 10. Мовою програмування для клієнтської частини було обрано JavaScript та бібліотеку React. Серверна частина написана мовою програмування C# з використанням фреймворку ASP.NET Core.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Для запуску застосунку, використовуючи Docker, потрібно спочатку з директорії проекту виконати команду “docker build” для побудови образу та “docker run” для запуску контейнера.

Запуск з операційної системи можна виконати або використовуючи вбудовані засоби середовища розробки Microsoft Visual Studio, або з командного рядка, виконавши команду “dotnet run” з директорії, де розміщений веб-застосунок.

ВХІДНІ І ВИХІДНІ ДАНІ

Вхідні дані описують задачу гідроакустики, яку потрібно вирішити. Користувач вводить дані у форму у вигляді тексту. На сервер вхідні дані відправляються у форматі JSON, де десеріалізуються в класи, написані мовою програмування C#.

Вихідні дані клієнту сервер повертає у форматі JSON, де вони відображаються у вигляді графіків та таблиць. Таблиці можна експортувати у форматі *.csv, *.xlsx, JSON, XML, а графіки в форматі *.png.